Mining Large Multi-Aspect Data: Algorithms and Applications

Evangelos E. Papalexakis

CMU-CS-16-124

August 12, 2016

Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Research was supported by the National Science Foundation Grant No. IIS-1247489, CNS-1314632, IIS-1408924, the Defense Threat Reduction Agency under contract No. HDTRA1-10-1-0120, the Army Research Office (ARO) and Defense Advanced Research Projects Agency (DARPA) under Contract Number W911NF-11-C-0088, the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-09-2-0053, the National Institute of Health under grant number R01GM108339, and Google. I would also like to thank the Open Cloud Consortium (OCC) and the Open Science Data Cloud (OSDC). The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: data mining, multi-aspect, multi-modal, matrix, tensor, tensor decomposition, tensor factorization, PARAFAC, Tucker, CORCONDIA, scalability, unsupervised learning, unsupervised analysis, exploratory analysis, brain data analysis, social network analysis, web search,PARCUBE,TURBO-SMT,PARACOMP,AUTOTEN

In loving memory of my father.

Abstract

What does a person's brain activity look like when they read the word apple? How does it differ from the activity of the same (or even a different person) when reading about an airplane? How can we identify parts of the human brain that are active for different semantic concepts? On a seemingly unrelated setting, how can we model and mine the knowledge on web (e.g., subject-verb-object triplets), in order to find hidden emerging patterns? Our proposed answer to both problems (and many more) is through bridging signal processing and large-scale multi-aspect data mining.

Specifically, language in the brain, along with many other real-word processes and phenomena, have different aspects, such as the various semantic stimuli of the brain activity (apple or airplane), the particular person whose activity we analyze, and the measurement technique. In the above example, the brain regions with high activation for "apple" will likely differ from the ones for "airplane". Nevertheless, each aspect of the activity is a signal of the same underlying physical phenomenon: language understanding in the human brain. Taking into account all aspects of brain activity results in more accurate models that can drive scientific discovery (e.g, identifying semantically coherent brain regions).

In addition to the above *Neurosemantics* application, multi-aspect data appear in numerous scenarios such as mining knowledge on the web, where different aspects in the data include entities in a knowledge base and the links between them or search engine results for those entities, and multi-aspect graph mining, with the example of multi-view social networks, where we observe social interactions of people under different means of communication, and we use all aspects of the communication to extract communities more accurately.

The main thesis of our work is that many real-world problems, such as the aforementioned, benefit from jointly modeling and analyzing the multi-aspect data associated with the underlying phenomenon we seek to uncover. In this thesis we develop scalable and interpretable algorithms for mining big multi-aspect data, with emphasis on *tensor decomposition*. We present algorithmic advances on scaling up and parallelizing tensor decomposition and assessing the quality of its results, that have enabled the analysis of multi-aspect data that the state-of-the-art could not support. Indicatively, our proposed methods speed up the state-of-the-art by up to two orders of magnitude, and are able to assess the quality for 100 times larger tensors. Furthermore, we present results on multi-aspect data applications focusing on *Neurosemantics* and *Social Networks and the Web*, demonstrating the effectiveness of multi-aspect modeling and mining. We conclude with our future vision on bridging Signal Processing and Data Science for real-world applications.

Acknowledgements

This thesis would have never existed if it hadn't been for quite a few great people that are in my life and have supported, inspired, and influenced me and my work. First and foremost, I am greatly indebted to my advisor Christos Faloutsos who has been the best mentor, advisor, and friend I could have asked for my PhD studies. We first met at ICDM 2010 in Sydney, even before I started my PhD, and little did I know how great of a role he would play in my life. He has been there, every step of the way, and literally any hour of the day, staying up before every single deadline, offering advice and support until the last minute! He has tirelessly shaped me into a (hopefully) better researcher and academic in general, and his support and advice especially during the last year of the job search have been invaluable. In addition to an advisor and a mentor, he is also a great friend; countless times I would enter a meeting stressed, and by the end of the meeting I would exit laughing loudly with all the negativity lifted. I will be forever grateful to Christos for all the opportunities and freedom he gave me generously: even as early as my first year, as a token of gratitude for helping out with a grant proposal, he paid for me to go to CIKM 2012 (without having any paper there, whatsoever) which "happened" to be located in Maui, Hawaii (and after that, he empowered me to travel literally all around the world, talking about my research, but that is a whole story on its own...). I could spend many pages expressing my gratitude to Christos, but in the interest of space I will just say that he is the greatest advisor who knows best how to inspire, nurture, and grow a student into a well-rounded academic citizen, and above all he is an amazing person.

None of what I have done so far would have happened if another great person, Nikos Sidiropoulos, had not been in my life. We started working together back at the Technical University of Crete for my undergraduate thesis, and subsequently my masters thesis. He is the one who first kindled my passion for research (and tensors!), and he is the one who enabled my first steps in academia, supporting my trip to ICDM 2010 (despite the economic hardships that Greek universities had started facing at the time) that, in retrospect might have been the most important milestone in my early academic life. He has always been an amazing mentor and a friend, and I am truly grateful for everything he has given me.

I would also like to thank the rest of my thesis committee, Tom Mitchell and Jeff Schneider. Tom is a role model to me both as an academic and as a person, and I feel extremely humbled and privileged to have been able to work with him (as a sidenote, the weekly meetings with Christos, Nikos, and Tom have been by far my favorite time of the week). I would especially like to thank Jeff for his amazing feedback for my thesis, which has given me food for thought for years to come.

In addition to my CMU activities, I was fortunate to work with great people in the industry. I am forever grateful to Rakesh Agrawal who offered me absolute freedom and countless thought-provoking hours of discussions during my 2014 internship at MSR, and is a fantastic mentor and subsequently collaborator, Ariel Fuxman, who offered me my first industry internship at MSR, Alon Halevy and Steven Whang who made my summer of 2015 at Google extremely enjoyable and definitely indulged my coffee drinking habits, and Stelios Paparizos who may have not been an official internship host, but has been a good friend and mentor.

I am blessed with many academic siblings from the Database group at CMU. I especially would like to thank Alex Beutel (my official conference roommate, my designated "group-therapy" coffee co-drinker, and my academic "little brother", just to name a few titles for this awesome guy), Danai Koutra (who has witnessed me multiple times order an "excessive amount" of espresso shots in a single beverage, has helped me beat a time-sharing company in Hawaii in exchange for a snorkelling trip, has cheered me up on numerous occasions, and who thinks that Alex and I will not play the Beijing videos to a venue of our choosing when she least expects it), Leman Akoglu (my sweet big sister who is always there for me). I would also like to thank my other older academic siblings: Aditya Prakash, Polo Chau, U Kang, Jimeng Sun, Spiros Papadimitriou, Hanghang Tong, Jure Leskovec, Tim Pan, Lei Li, and Fan Guo for all their advice and mentorship, the younger ones: Neil Shah, Miguel Araujo, Hemank Lamba, Hyun-Ah Song, Bryan Hooi, and Dhivya Eswaran, as well as all the amazing visitors we have had over the years, and especially: Yasuko Matsubara, Yasushi Sakurai, SunHee Kim, Stephan Günnemann, Meng Jiang, Gonzalo Mateos, and Srijan Kumar.

Outside of the Database group, I would like to thank all of my friends at CMU: my two awefficemates Anna Gommerstadt and John Dickerson (who bring the *awesome* to the *aweffice*), my SCS friends and especially the two Davids (Naylor and Witmer), Dougal Sutherland, Richard Wang, Nico Feltman, Jay-Yoon Lee, Dana and Yair Movshovitz-Attias, Walter Lasecki, and many more, Andy Pavlo for his advice especially during my job search, Christina Cowan, Catherine Copetas and my Greek gang in Pittsburgh, especially Elina Karnezi (aka the good cat) for being an amazing friend and being always there for me, Eleni Katrini, Antonis Tasoglou, Anthony Platanios, Nick Katsipoulakis, Nikos Lappas, and Maria Tomprou.

CMU would be a totally different place for me if it weren't for Marilyn Walgora. Anything I write here will never do justice to how much of an amazing person Marilyn is! In addition to the fact that she is probably the most efficient administrative assistant that ever existed, she rightfully deserves the title of my "American Mom", having supported me during these five years, and I am eternally grateful to her. I would also like to thank Deb Cavlovich who has always been there, making sure my studies at CMU run smoothly.

I would like to thank all of my collaborators and academic friends that I have not mentioned yet. I especially would like to thank Kostas Pelechrinis who in addition to a great collaborator is also a very good friend, Tina Eliassi-Rad who is an amazing mentor, collaborator, and honorary academic cousin to me (as well as Branden Fitelson, who is not a collaborator *yet*, but is a great friend and mentor), Tanya Berger-Wolf, Michalis Faloutsos, Nitesh Chawla, Partha Pratim Talukdar, Seza Dogruoz, Brian Murphy, Evimaria Terzi, Behzad Golshan, Alona Fyshe, Dino Ienco, Eric Mao, Kejun Huang, Xiao Fu, Panos Adamopoulos, Vilma Georgia Todri, Panos Ipeirotis, Marios Kokkodis, Marialena Kyriakidi, Orestis Polychroniou Alex Labrinidis, Panos Chrysanthis, my conference buddies Xiao Yu, Jessie Zhenhui Li, and Airalee Studley and my Chemometrics/Tensor gang: Rasmus Bro and Evrim Acar. A special thank you goes to everyone who has hosted me as an invited speaker: Mohammad Hammoud at CMU Qatar, David Lo and Hady Lauw at Singapore Management University, Sofus Macskassy at Facebook, Peng Cui at Tsinghua, Ping

Luo at Chinese Academy of Sciences, and Yu Zheng at Microsoft Research Asia.

Before my life at CMU, I was at the Technical University of Crete (TUC) where I met some incredible people. I would like to especially thank my good friends Dimitra Tzanetatou, Angelica Topalidou-Kyniazopoulou, Foivos Gypas, Kostas Apostolou, George Dalaperas, Pavlos Malakonakis, Alexander Sklikas and many more who have been there for me for over a decade, as well as my professors back there, and especially Athanasios Liavas, Minos Garofalakis, George Karystinos, Michail Lagoudakis, Antonis Deligiannakis, and Aggelos Bletsas.

I would like to especially thank Nini Liao who has been my closest and dearest friend in Pittsburgh, with whom I have shared countless moments of joy. Last but not least, I would like to thank my family and especially my parents, Vasiliki Galliaki and Evangelos Papalexakis (*yes*, same name!), for their selfless sacrifice, love, and support. Anything that is good in my life is due to them. Unfortunately, my father passed away during the last year of my PhD. It pains me greatly that he cannot see the culmination of my academic studies, something that he had been fighting for, been proud of, and anticipating for the last 28 years of his life. I can only hope that I lived up to his dream and I dedicate this thesis to him as a tiny token of how much I loved him, and how much my Mom (whom I love unconditionally) and I miss him.

Contents

Chapter 1

Introduction

1.1 Overview of Contributions and Impact

What does a social graph between people who *call* each other look like? How does it differ from one where people *instant-message* or *e-mail* each other? Social interactions, along with many other real-word processes and phenomena, have different *aspects*, such as the means of communication. In the above example, the activity of people calling each other will likely differ from the activity of people instant-messaging each other. Nevertheless, each aspect of the interaction is a signature of the same underlying social phenomenon: formation of social ties and communities. Taking into account all aspects of social interaction results in more accurate social models (e.g, communities).

Our main thesis is that many real-world problems benefit from jointly modeling and analyzing the multi-aspect data associated with the underlying phenomenon we seek to uncover.

The overarching theme of our work is focusing on scalable and interpretable algorithms for mining big multi-aspect data by **bridging Signal Processing and Data Science for real-world applications**.

This thesis is broken down to *algorithms* with contributions in tensor analysis and multi-aspect data mining *applications*.

1.1.1 Algorithms

The primary computational tool in this thesis is *tensor decomposition*. Tensors are multidimensional matrices, where each *aspect* of the data is mapped to one of the dimensions or *modes*. Chapter 2 is a comprehensive survey of existing tensor models, algorithms, and applications, for the interested reader. In order to analyze a tensor, we compute a decomposition or factorization (henceforth we use the terms interchangeably), which gives a low-dimensional *embedding* of all the aspects. We primarily focus on the Canonical or PARAFAC decomposition, which decomposes the tensor into a sum of outer products

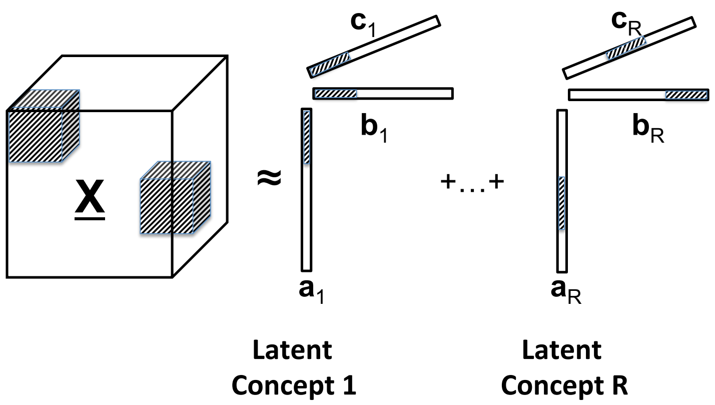


Figure 1.1: Canonical or PARAFAC decomposition into sum of R rank-one components. Each component is a *latent concept* or a *co-cluster*.

of *latent factors* (see also Figure 1.1):

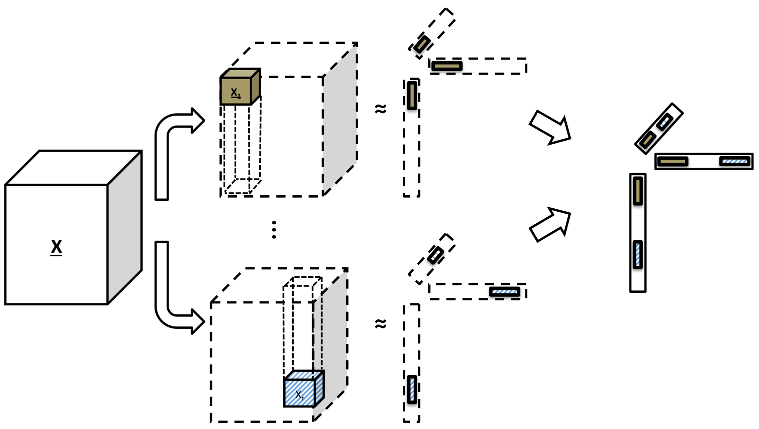
$$\underline{\mathbf{X}} \approx \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r$$

where $\circ$ denotes outer product, i.e., $[\mathbf{a} \circ \mathbf{b} \circ \mathbf{c}](i,j,k) = \mathbf{a}(i)\mathbf{b}(j)\mathbf{c}(k)$. Informally, each latent factor is a co-cluster of the aspects of the tensor. The advantages of this decomposition over other existing ones are interpretability (each factor corresponds to a co-cluster), and strong uniqueness guarantees for the latent factors.Tensor decompositions are very powerful tools, with numerous applications (see details in Chapters 2 and 3). There is increasing interest in their application to Big Data problems, both from academia and industry. However, algorithmically, there exist challenges which limit their applicability to real-world, big data problems, pertaining to scalability and quality assessment of the results. *Below we outline how this thesis addresses those challenges, towards a broad adoption of tensor decompositions in big data science.*

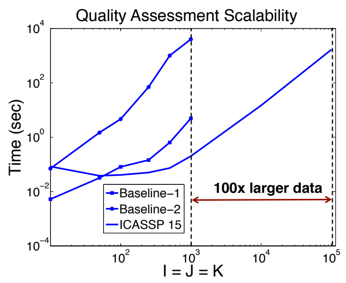
1.1.1.1 Parallel, and Scalable Tensor Decompositions

Consider a multi-aspect tensor dataset that is too big to fit in the main memory of a single machine. The data may have large "ambient" dimension (e.g., a social network can have billions of users), however the observed interactions are very sparse, resulting in extremely sparse data. This data sparsity can be exploited for efficiency. In Chapter 4 we formalize the above statement by proposing the concept of a **triple-sparse algorithm** where 1) the input data are sparse, 2) the intermediate data that the algorithm is manipulating or creating are sparse, and 3) the output is sparse. Sparsity in the intermediate data is crucial for scalability. Sparsity in the results is a great advantage, both in terms of storage but most importantly in terms of interpretability, since sparse models are easier for humans to inspect. *None of the existing state of the art algorithms, before [PFS12] when the peer-reviewed version of Chapter 4 first appeared, fulfilled all three requirements for sparsity.*

In Chapter 4 we propose ParCube, the first triple-sparse, parallel algorithm for tensor decomposition. Figure 1.2(a) depicts a high-level overview of ParCube. Suppose that we computed a weight of how important every row, column, and "fiber" (the third mode index) of the tensor is. Given that weight, ParCube takes biased samples of rows, columns, and fibers, extracting a small tensor from the full data. This is done repeatedly with each sub-tensor effectively explores different parts of the data. Subsequently, ParCube decomposes all those sub-tensors *in parallel* generating partial results. Finally, ParCube merges the partial results ensuring that partial results corresponding to the same latent component are merged together. The power behind ParCube is that, even though the tensor itself might not fit in memory, we can choose the sub-tensors appropriately so that they fit in memory, and we can compensate by extracting many independent sub-tensors. ParCube converges to the same level of sparsity as [PSB13] (the first tensor decomposition with latent sparsity) and furthermore ParCube's approximation error converges to the one of the full decomposition (in cases where we are able to run the full decomposition). This demonstrates that ParCube's sparsity maintains the useful information in the data. In Chapter 5, we extend the idea of ParCube, introducing Turbo-SMT, for the case of Coupled Matrix-Tensor Factorization (CMTF), where a tensor and a matrix share one of the aspects of the data, achieving up to 200 times faster execution with comparable accuracy to the baseline, on a single machine [PMS+14]. Subsequently, in Chapter 6 we



(a) **The main idea behind ParCube** (Chapter 4) : Using biased sampling, extract small representative sub-sampled tensors, decompose them in parallel, and carefully merge the final results into a set of sparse latent factors.



(b) **Scalable Quality Assessment** (Chapter 7): Computing the decomposition quality for tensors for two orders of magnitude larger tensor than the state of the art (I, J, K are the tensor dimensions).

Figure 1.2: Overview of our algorithmic results.

propose Paracomp, a novel parallel architecture for tensor decomposition in similar spirit as ParCube. Instead of sampling, Paracomp uses random projections to *compress* the original tensor to multiple smaller tensors. Thanks to compression, in Chapter 6 we prove that Paracomp can guarantee the *identifiability* of the results. This is a very strong guarantee on the correctness and quality of the result.

In addition to Chapters 4, 5, and 6 which introduce a novel paradigm for parallelizing and scaling up tensor decomposition, in [KPHF12] we developed the first scalable algorithm for tensor decompositions on Hadoop which was able to decompose problems larger by at least **two orders of magnitude** than the state of the art. Subsequently [BKP$^+$14] we developed a Distributed Stochastic Gradient Descent method for Hadoop that scales to billions of parameters. We provide a description of those methods in Section 3.4 of Chapter 3.

1.1.1.2 Unsupervised Quality Assessment of Tensor Decompositions

Real-world exploratory analysis of multi-aspect data is, to a great extent, *unsupervised*. Obtaining ground truth is a very expensive and slow process, or in the worst case impossible; for instance, in *Neurosemantics* where we research how language is represented in the brain, most of the subject matter is uncharted territory and our analysis drives the exploration. Nevertheless, we would like to assess the quality of our results in absence of ground truth. There is a very effective heuristic in Signal Processing and Chemometrics literature by the name of "Core Consistency Diagnostic" (see Chapter 7) which assigns a "quality" number to a given tensor decomposition and gives information about the data being inappropriate for such analysis, or the number of latent factors being incorrect. However, this diagnostic has been specifically designed for fully dense and small datasets, and is not able to scale to large and sparse data. In Chapter 7, exploiting sparsity, we introduce a *provably exact algorithm* that operates on at least **two orders of magnitude larger data** than the state of the art (as shown in Figure 1.2(b)), which enables quality assessment on large real datasets for the first time. Subsequently, in Chapter 8, we extend the Core Consistency Diagnostic under the KL-Divergence loss, which assumes a Poisson distribution for the data. It has been shown that such an assumption is more accurate for sparse, count data, the types of which we are mostly dealing with. In addition, we conduct a large-scale study of the decomposition quality of many real datasets, which to the best of our knowledge is the first of its kind. Finally, we propose AutoTen, a comprehensive unsupervised and automatic tensor mining method that provides quality assessment of the results. AutoTen outperforms state-of-the-art approaches in determining the number of components in a tensor, which is an extremely hard problem, and of utmost importance to solve for real-world applications.

Impact - Algorithms

- ParCube [PFS12] is the most cited paper of ECML-PKDD 2012 with over 60 citations at the time of writing, whereas the median number of citations for ECML-PKDD 2012 is 5. Additionally, ParCube has already been downloaded over 125 times by universities and organizations from 23 countries.
- ParCube has been featured in an article by the Army Research Laboratory *Six Potential Game-Changers in Cyber Security: Towards Priorities in Cyber Science and Engineering* [KSM15] which was also presented as a keynote talk at the NATO

Symposium on Cyber Security Science and Engineering 2014.
- TURBO-SMT [PMS+14] was selected as one of the best papers of SDM 2014, and appeared in a special issue of the Statistical Analysis and Data Mining journal [PMS+16].
- PARACOMP [SPF14] has appeared in the prestigious IEEE Signal Processing Magazine.
- Our SDM 2016 paper introducing AUTOTEN [Pap16] won the NSF Travel Award and Best Student Paper Award.

1.1.2 Applications

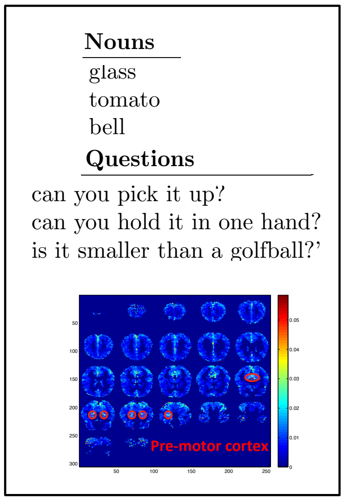
The two main application areas of this thesis are: 1) *Neurosemantics* and 2) *Social Networks and the Web*

1.1.2.1 Neurosemantics

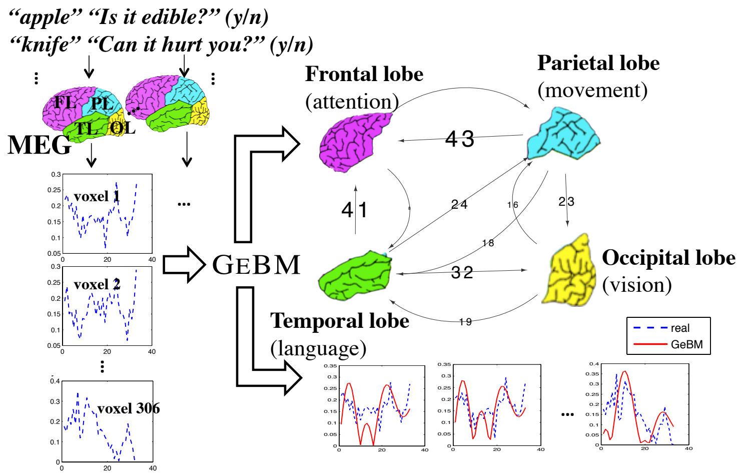
How is knowledge represented in the human brain? Which regions have high activity and information flow, when a concept such as "food" is shown to a human subject? Do all human subjects' brains behave similarly in this context? Consider the following experimental setting, where multiple human subjects are shown a set of concrete English nouns (e.g. "dog", "tomato" etc), and we measure each person's brain activity using various techniques (e.g, fMRI or MEG). In this experiments, human subjects, semantic stimuli (i.e., the nouns), and measurement methods are all different aspects of the same underlying phenomenon: the mechanisms that the brain uses to process language.

In Chapter 9 we seek to identify coherent regions of the brain that are activated for a semantically coherent set of stimuli. To that end we combine fMRI measurements with semantic features (in the form of simple questions, such as *Can you pick it up?*) for the same set of nouns, which provide useful information to the decomposition which might be missing from the fMRI data, as well as constitute a human readable description of the semantic context of each latent group. A very exciting example of our results can be seen in Figure 1.3(a), where all the nouns in the "cluster" are small objects, the corresponding questions reflect holding or picking such objects up, and most importantly, the brain region that was highly active for this set of nouns and questions was the *premotor cortex*, which is associated with holding or picking small items up. This result is **entirely unsupervised** and agrees with Neuroscience. This gives us confidence that the same technique can be used in more complex tasks and drive Neuroscientific discovery.

In a similar experimental setting, where the human subjects are also asked to answer a simple yes/no question about the noun they are reading, in Chapter 10 we seek to discover the *functional connectivity* of the brain for the particular task. Functional connectivity is an information flow graph between different regions of the brain, indicating high degree of interaction between (not necessarily physically connected) regions while the person is reading the noun and answering the question. [EFS+] we propose GEBM, a novel model for the functional connectivity which views the brain as a *control system* and we propose a sparse system identification algorithm which solves the model. Figure



(a) **Unsupervised Discovery of Semantically Coherent Brain Regions** (Chapter 9): The pre-motor cortex, having high activation here, is associated with motions such as holding and picking small items up.



(b) **Estimating Brain Functional Connectivity** (Chapter 10): Given MEG measurements (time series of the magnetic activity of a particular region of the brain), GeBM learns a graph between physical brain regions. Furthermore, GeBM simulates the real MEG activity very accurately.

Figure 1.3: Overview of results of the Neurosemantics application.

1.3(b) shows an overview of GeBM: given MEG measurements (time series of the magnetic activity of a particular region of the brain), we learn a model that describes a graph between physical brain regions, and simulates real MEG activity very accurately.

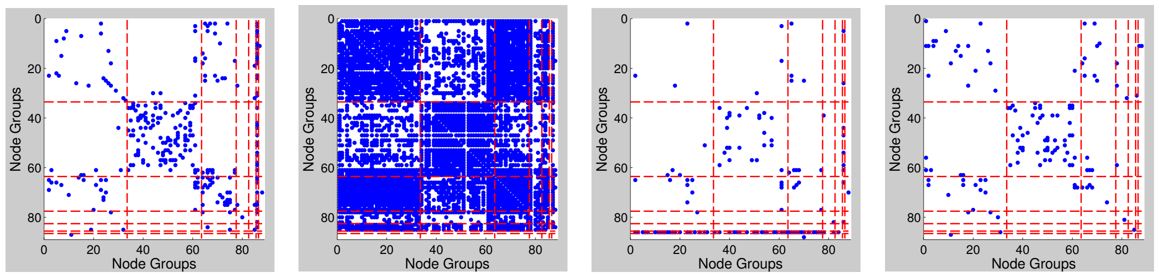
Impact - Neurosemantics

- GeBM [EFS+] is taught in class `CptS 595` at Washington State University.
- Our work in [PMS+14] (appearing in Chapter 9) was selected as one of the best papers of SDM 2014

1.1.2.2 Social Networks and the Web

This second class of applications focuses on extracting useful knowledge from social networks and the web by exploiting the multiple aspects of the data and using tensors. In this thesis we include two representative pieces of a long line of work in this thrust (Chapters 11 and 12), however, in the following lines we also briefly cover a broader set of works.

In Chapter 11 we introduce GraphFuse, a tensor based community detection algorithm which uses different aspects of social interaction and outperforms state of the art in community extraction accuracy. Figure 1.4 shows GraphFuse at work, identifying communities in `Reality Mining`, a real dataset of multi-aspect interactions between



(a) `calls` (b) `proximity` (c) `sms` (d) `friends`

Figure 1.4: **Consistent communities across views** (Chapter 11): Results on the four views of the `Reality Mining` multi-graph. Red dashed lines outline the clustering found by GRAPHFUSE.

students and faculty at MIT. The communities are consistent across aspects and agree with ground truth. Another aspect of a social network is *time*. In [APG$^+$14] we introduce COM2, a tensor based *temporal community* detection algorithm, which identifies social communities and their behavior over time. This work is not included as a chapter of this thesis, however, the key results are summarized in Chapter 3, Section 3.3.1. In [ED15] we consider *language* as another aspect, where we identify topical and temporal communities in a discussion forum of Turkish immigrants in the Netherlands. We observe interesting variations of how bilingual or monolingual each community is based on the topic of discussion (which was automatically extracted) e.g., sports were discussed primarily in Turkish while trendy matters were discussed in Dutch. Adding to language, *location* is one aspect that has become extremely pervasive recently, with virtually all smartphones allowing for the geolocation of the user, and most on-line social platforms enabling the location tagging of a user's activity. In [PPF15] we explore data from Foursquare, identifying spatial and temporal patterns of users' physical and social activity. In addition to the aforementioned works, we include more results on analyzing time-evolving social networks in Section 4.4 of Chapter 4. Our work is not necessarily restricted to social networks, e.g., we have also analyzed multi-aspect computer networks, detecting *anomalies* and *network attacks* [MWP$^+$14].

In addition to Social Networks, in this part we also focus on extracting knowledge from the Web, which also has multiple aspects: real-world entities such as *Barack Obama* and *USA* are usually linked in multiple ways, e.g., *is president of*, *was born in*, and *lives in*. Modeling those multi-aspect relations a tensor, and computing a low rank decomposition of the data, results in *embeddings* of those entities in a lower dimension, which can help discover semantically and contextually similar entities, as well as discover missing links. In [PFS12] we discover semantically similar noun-phrases in a Knowledge Base coming from the *Read the Web* project at CMU: `http://rtw.ml.cmu.edu/rtw/`. Results from this analysis are included in Section 4.4 of Chapter 4.

Yet another aspect of an real-world entity on the web is the set of *search engine results* for that entity, which is the biased view of each search engine, as a result of their crawling, indexing, ranking, and potential personalization algorithms, for that query. In [AGP15a] we introduce TensorCompare, a tool which measures the overlap in the results of different search engines. We conduct a case study on Google and Bing, finding high overlap. Given this high overlap, how can we use different signals, potentially coming from social media, in order to provide diverse and useful results? In [AGP15b] we follow-up designing a Twitter based web search engine, using tweet popularity as a ranking function, which does exactly that. This result has huge potential for the future of web search, paving the way for the use of social signals in the determination and ranking of results. We combine the results of those two papers in Chapter 12.

Impact - Social Networks and the Web

- Com2 [APG+14] won the best student paper award at PAKDD 2014.
- GraphFuse [PAI13] has been downloaded over 100 times from 21 countries.
- Our work in [MWP+14] (which is based on our scalable tensor analysis tools) is deployed by the Institute for Information Industry in Taiwan, detecting real network intrusion attempts.
- Our work in [AGP15a] was selected to appear in a special issue of the Journal of Web Science [AGP+16], as one of the best papers in the Web Science Track of WWW 2015.

1.2 Thesis Organization

The rest of this document is organized as follows. Chapter 2 introduces the necessary notation and tensor decompositions used throughout this thesis. Chapter 3 is a detailed and concise introduction to tensor decompositions from a practitioner's point of view, covering different decomposition models that are referenced throughout this thesis, as well as data mining applications and scalable algorithms, providing an overview of the algorithms proposed in this thesis, in comparison to existing methods and subsequent work that followed. Chapter 3 is *not* essential for following the concepts behind this thesis, however, serves as a reference to the interested reader who wishes to delve deeper into tensor decompositions.

For Parts I, II, III, and IV, each chapter is preceded by an executive summary in blue frame.

Part I contains three chapters on scalable tensor decomposition algorithms (Chapters 4, 5, and 6), and Part II contains two chapters on scalable quality assessment of the decomposition results (Chapters 7 and 8).

Each application class has its own part: Part III contains our work on *Neurosemantics*, where Chapter 9 identifies semantically coherent regions of the brain, and Chapter 10 uses control theory to model the functional connectivity of the brain. Part IV demonstrates our work on the *Social Networks and the Web* application, with Chapter 11 exploring the effect of multiple views of a social network to community detection, and Chapter 12 outlining our results on comparing search engines using tensors, and using social media as an alternative to traditional web search.

Finally, in Chapter 13 we draw the collective conclusion of this thesis, highlighting its broader impact, and in Chapter 14 we sketch future research directions in tensor algorithms and applications, as well as our future vision on bridging Signal Processing and Data Science for real-world applications.

Chapter based on material to be published in [PFS].

Chapter 2

Background

In this Chapter we provide the necessary notation and definitions of tensor decompositions used throughout this thesis.

2.1 Introduction

Tensors are multi-dimensional extensions of matrices. Because of their ability to express multi-modal or multi-aspect data, they are very powerful tools in applications that inherently create such data. For instance, in on-line social networks, people tend to interact with each other in a variety of ways: they message each other, they post on each others' pages, and so on. All these different means of interaction are different aspects of the same social network of people, and can be modeled as a three-mode tensor, a "data-cube", of (user, user, means of interaction). Given this tensor, there exists a rich variety of tools called tensor decompositions or factorizations, which are able to extract meaningful, latent structure in the data. In this Chapter, we describe the tensor decompositions that we use throughout this thesis. In Section 2.2 we provide the necessary notation, and in Section 2.3 we introduce PARAFAC, TUCKER and Coupled Matrix-Tensor Factorization (CMTF).

2.2 Preliminary Definitions & Notation

In this section we provide a few necessary definitions and describe our notation. Table 2.1 summarizes our notation throughout this thesis. The notation on Table 2.1 is universal throughout the entire thesis, however, in some chapters we introduce additional notation.

Definition 1. Outer product: Given two vectors $\mathbf{a} \in \mathbb{R}^{I \times 1}$ and $\mathbf{b} \in \mathbb{R}^{I \times 1}$ their outer product is an $I \times J$ matrix and is denoted as $\mathbf{a} \circ \mathbf{b}$. Its (i, j)-th entry is equal to $\mathbf{a}(i)\mathbf{b}(j)$. This definition can be extended to an arbitrary number of vectors.

Definition 2. Kronecker product: Given two matrices $\mathbf{A} \in \mathbb{R}^{I \times J}$ and $\mathbf{B} \in \mathbb{R}^{K \times L}$, their

Symbol	**Definition**
$\underline{\mathbf{X}}, \mathbf{X}, \mathbf{x}, x$	Tensor, matrix, column vector, scalar
$\mathbb{R}$	The set of real numbers
$\mathbf{x} \in \mathbb{R}^{I \times 1}$	Definition of an I-dimensional vector (same for matrices and tensors)
$\circ$	Outer product
$vec(\ )$	Vectorization operator
$diag(\mathbf{A})$	Diagonal of matrix $\mathbf{A}$
$Diag(\mathbf{a})$	Diagonal matrix with a in the diagonal
$\otimes$	Kronecker product
$\odot$	Khatri-Rao product
$*\ \oslash$	Element-wise multiplication and division
$\times_n$	n-mode product
$\mathbf{X}_{(n)}$	n-mode matricization of tensor $\underline{\mathbf{X}}$
$\mathbf{A}^{-1}$	Inverse of $\mathbf{A}$
$\mathbf{A}^{\dagger}$	Moore-Penrose Pseudoinverse of $\mathbf{A}$
$D_{KL}(a\|\|b)$	KL-Divergence
$\|\mathbf{A}\|_F$	Frobenius norm
$\mathbf{x}(i)$	i-th entry of x (same for matrices and tensors)
$\mathbf{x}(\mathcal{I})$	Spans the elements of x corresponding to indices in set $\mathcal{I}$
$\mathbf{X}(:, i)$	Spans the entire i-th column of $\mathbf{X}$ (same for tensors)
$\mathbf{X}(i, :)$	Spans the entire i-th row of $\mathbf{X}$ (same for tensors)
reshape( )	Rearrange the entries of a given matrix or tensor to a given set of dimensions
numel( )	For an $I_1 \times I_2 \cdots \times I_N$ tensor, returns $\prod_{n=1}^{N} I_n$
MTTKRP	Matricized Tensor Times Khatri-Rao Product

Table 2.1: Table of symbols

Kronecker product is an $IK \times JL$ matrix equal to:

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} \mathbf{A}(1,1)\mathbf{B} & \cdots & \mathbf{A}(1,j)\mathbf{B} & \cdots & \mathbf{A}(1,J)\mathbf{B} \\ \vdots & \cdots & \vdots & \cdots & \vdots \\ \mathbf{A}(i,1)\mathbf{B} & \cdots & \mathbf{A}(i,j)\mathbf{B} & \cdots & \mathbf{A}(i,J)\mathbf{B} \\ \vdots & \cdots & \vdots & \cdots & \vdots \\ \mathbf{A}(I,1)\mathbf{B} & \cdots & \mathbf{A}(I,j)\mathbf{B} & \cdots & \mathbf{A}(I,J)\mathbf{B} \end{bmatrix}$$

Definition 3. Khatri-Rao product: Given two matrices $\mathbf{A}$ and $\mathbf{B}$, their Khatri-Rao product is defined as the column-wise Kronecker product:

$$\mathbf{A} \odot \mathbf{B} = \left[\mathbf{A}(:,1) \otimes \mathbf{B}(:,1) \quad \cdots \quad \mathbf{A}(:,j) \otimes \mathbf{B}(:,j) \quad \cdots \quad \mathbf{A}(:,J) \otimes \mathbf{B}(:,J)\right]$$

Definition 4. N-mode product: Given an N-mode tensor $\underline{\mathbf{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ and a matrix $\mathbf{A} \in \mathbb{R}^{I_n \times R}$, the n-mode product of $\underline{\mathbf{X}}$ and $\mathbf{A}$ is denoted as $\underline{\mathbf{Y}} = \underline{\mathbf{X}} \times_n \mathbf{A}$ where $\underline{\mathbf{Y}} \in \mathbb{R}^{I_1 \times \cdots I_{n-1} \times R \times I_{n+1} \times \cdots \times I_N}$, and

$$\underline{\mathbf{Y}}(i_1, \cdots, i_{n-1}, r, i_{n+1}, \cdots, i_n) = \sum_{j=1}^{I_n} \underline{\mathbf{X}}(i_1, \cdots, i_{n-1}, j, i_{n+1}, \cdots i_N)\mathbf{A}(j,r)$$

Definition 5. N-mode matricization: An N-mode tensor can be unfolded or matricized into a matrix in N ways, one for each mode. The n-mode matricization of $\underline{\mathbf{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is denoted as $\mathbf{X}_{(n)} \in \mathbb{R}^{I_n \times I_1 \cdots I_{n-1} I_{n+1} \cdots I_N}$ and is taken by keeping the n-th mode intact and concatenating the slices of the rest of the modes into a long matrix. Figure 2.1 shows an example of a three-mode tensor and its 1-mode matricization. Note that, as mentioned in [KB09], there exist various definitions of the order in which the slices are concatenated during the matricization, however, the end result is the same, as long as the order is consistent across different matricizations.

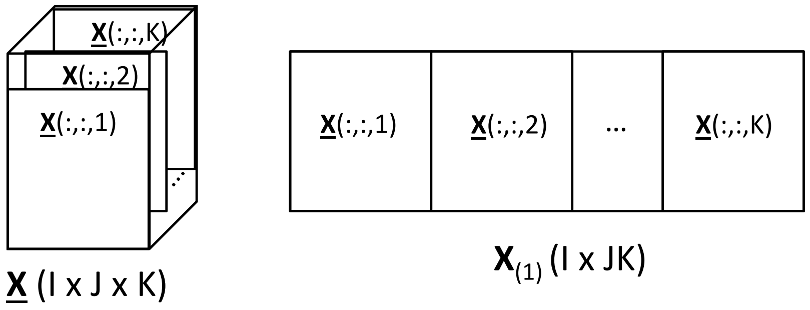


Figure 2.1: 1-mode matricization of $\underline{\mathbf{X}}$ to matrix $\mathbf{X}_{(1)}$.

Property 2.1. Vectorization and Khatri-Rao product:
property of the vec operator and the Khatri-Rao product [Bre78]

$$vec\left(\mathbf{A}\mathbf{D}\mathbf{B}^T\right) = \left(\mathbf{B} \odot \mathbf{A}\right) diag\left(\mathbf{D}\right)$$

2.3 Introduction to PARAFAC, TUCKER, and CMTF

2.3.1 PARAFAC Decomposition

The Polyadic, CP or PARAFAC decomposition (henceforth referred to as PARAFAC) was independently proposed by [Hit27], [CC70a] and[Har70]. In addition to those previous names, we also find this decomposition named *trilinear* (or multilinear in the case of a tensor with more than three modes). Given a three-mode tensor $\underline{\mathbf{X}} \in \mathbb{R}^{I \times J \times K}$ its PARAFAC decomposition is defined as the decomposition of the tensor into a sum of R rank-one tensors, or equivalently as the sum of R outer products of three vectors:

$$\underline{\mathbf{X}} \approx \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r \tag{2.1}$$

where $\mathbf{a}_r \in \mathbb{R}^{I \times 1}, \mathbf{b}_r \in \mathbb{R}^{J \times 1}, \mathbf{c}_r \in \mathbb{R}^{K \times 1}$, and $(\mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r)(i,j,k) = \mathbf{a}_r(i)\mathbf{b}_r(j)\mathbf{c}_r(k)$. When R is equal to the rank of the tensor, then Eq. 2.1 is exact equality, and when R is smaller, it is an approximation. As a result, the PARAFAC decomposition can reveal the rank of $\underline{\mathbf{X}}$ which is defined as the minimum R to perfectly reconstruct $\underline{\mathbf{X}}$ using the components of the decomposition. For compactness, the PARAFAC decomposition is usually denoted by the the factor matrices $\mathbf{A} \in \mathbb{R}^{I \times R}, \mathbf{B} \in \mathbb{R}^{J \times R}, \mathbf{C} \in \mathbb{R}^{K \times R}$, whose columns contain $\mathbf{a}_r$, $\mathbf{b}_r$, and $\mathbf{c}_r$ respectively. Figure 2.2 shows a pictorial representation of the decomposition. Often, we may assume that the columns of $\mathbf{A}, \mathbf{B}, \mathbf{C}$ are normalized, and in this case, each latent component is accompanied by a scalar λ_r which absorbs the scaling for $\mathbf{a}_r$, $\mathbf{b}_r$, and $\mathbf{c}_r$.

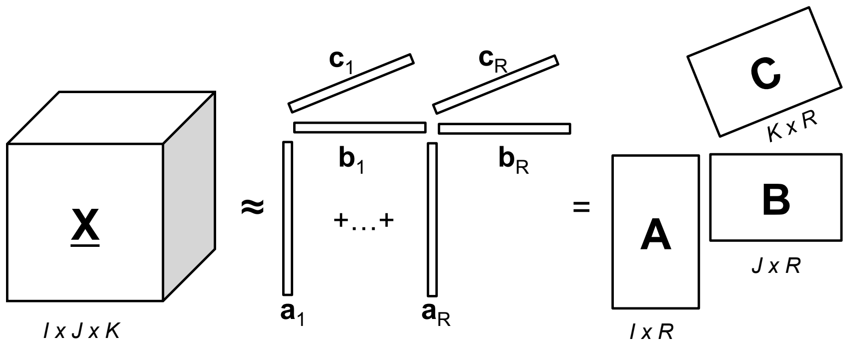


Figure 2.2: The PARAFAC decomposition.

In the case of an N-mode tensor, we can readily extend the PARAFAC decomposition by adding a set of factor vectors per additional mode. When $N > 3$, for notational simplicity we denote the factors as $\mathbf{A}^{(N)}$, and we write

$$\underline{\mathbf{X}} \approx \sum_{r=1}^{R} \mathbf{a}_r^{(1)} \circ \mathbf{a}_r^{(2)} \circ \cdots \circ \mathbf{a}_r^{(N)}$$

PARAFAC is one of the most highly used tensor decompositions, in part due to its *ease of interpretation*. Each rank-one component of the decomposition serves as a latent "concept" or cluster in the data, and the factor vectors for that component serve as the soft memberships of each row, column, and fiber of the tensor, to that latent cluster. As one of the papers that introduces PARAFAC [Har70] states, this is an *explanatory* model. As such, it is suitable for exploratory data mining, as we will also see in the applications section (Sec. 3.3)

In addition to admitting a very intuitive interpretation, PARAFAC enjoys important theoretical properties. In particular, it is shown that PARAFAC is essentially unique, under very mild conditions, up to scaling and permutation of the R components [Kru77, SB00, tBS02, JS04, StBDL06, CO12].

This is in sharp contrast to Matrix Factorization where we generally have

$$\mathbf{X} \approx \sum_{r=1}^{R} \mathbf{a}_r \mathbf{b}_r^T \approx \mathbf{A}\mathbf{B}^T = \mathbf{A}\mathbf{Q}\mathbf{Q}^{-1}\mathbf{B}^T = \tilde{\mathbf{A}}\tilde{\mathbf{B}}^T$$

for any invertible $\mathbf{Q}$, i.e., there exist multiple such decompositions of $\mathbf{X}$ that yield the same approximation error.

In practice, this means that in applications where not only care about approximating the original data well, but we also care about *interpreting* the latent factors, PARAFAC guarantees that the returned factors are the ones that generated the variation in the data that we seek to interpret and understand.

Practitioner's Guide

Strengths: Explanatory model, essentially unique under mild conditions, and easy to interpret.

Weaknesses: Hard to determine the appropriate rank and the global minimum.

When to use?: Extracting latent factors for interpretation, explanatory clustering, and etc.

2.3.2 TUCKER Decomposition

Another extremely popular decomposition is the TUCKER Decomposition, originally proposed by [Tuc66]. In fact, [Tuc66] proposed three different models, but we are going to focus on the third, also known as Tucker-3 (henceforth referred to as TUCKER). [KB09] provides an excellent overview of Tucker-1 and Tucker-2. The TUCKER decomposition was further popularized by [DLDMV00], wherein the authors pose the decomposition as an extension of the Singular Value Decomposition for tensors, naming the decomposition Higher-Order Singular Value Decomposition (HOSVD). Subsequently, [HDLL08] show that HOSVD can be used as K-means clustering for higher-order data, popularizing TUCKER models in the data mining community.

In contrast to PARAFAC, TUCKER decomposes a three mode tensor $\underline{\mathbf{X}} \in \mathbb{R}^{I \times J \times K}$ into three factor matrices $\mathbf{U}_1 \in \mathbb{R}^{I \times R_1}$, $\mathbf{U}_2 \in \mathbb{R}^{J \times R_2}$, $\mathbf{U}_3 \in \mathbb{R}^{K \times R_3}$, which are also multiplied by a core tensor $\underline{\mathbf{G}}$.

$$\underline{\mathbf{X}} \approx \underline{\mathbf{G}} \times_1 \mathbf{U}_1 \times_2 \mathbf{U}_3 \times_3 \mathbf{U}_3 \tag{2.2}$$

Alternatively, the decomposition can be written element-wise as:

$$\underline{\mathbf{X}}(i,j,k) \approx \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \sum_{r_3=1}^{R_3} \underline{\mathbf{G}}(r_1,r_2,r_3) \mathbf{U}_1(i,r_2) \mathbf{U}_2(j,r_2) \mathbf{U}_3(k,r_3)$$

A pictorial example of TUCKER is shown in Figure 2.3. The core tensor captures interactions between the columns of $\mathbf{U}_i$. Furthermore, we can assume that $\mathbf{U}_i$ are orthogonal matrices, and as stated in [AABB⁺07], they reflect the main subspace variation in each mode assuming a multi-linear structure.

TUCKER is generally non-unique, in contrast to PARAFAC. As in the matrix factorization case, we can rotate the factors without affecting the reconstruction error. However, TUCKER yields a good low rank approximation of a tensor in terms of squared error. This approximation can be also seen as compression, since the core tensor $\underline{\mathbf{G}}$ is the best compression of the original tensor with respect to squared error, a property that we will revisit in Section 3.4.1 when discussing ways of speeding up the PARAFAC decomposition.

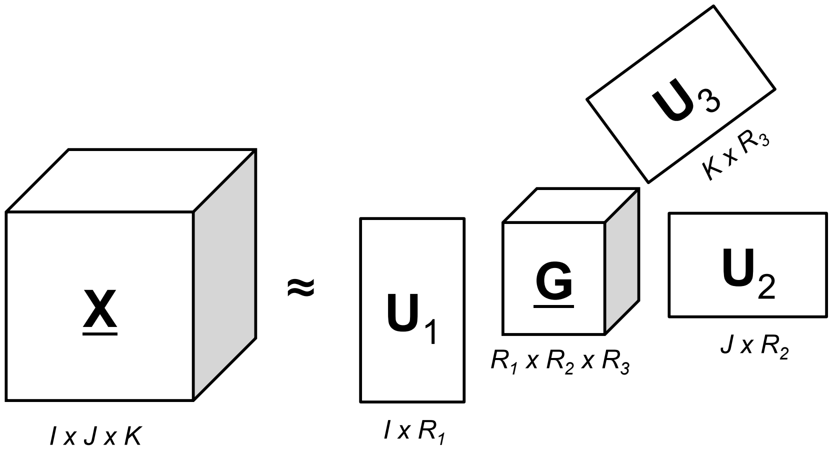


Figure 2.3: The TUCKER Decomposition

A very interesting observation is that PARAFAC can be written as a restricted TUCKER model. In particular, we can view PARAFAC as a TUCKER model where $R_1 = R_2 = R_3 = R$ (allowing R to be as high as $\min(IJ, JK, IK)$) and the core $\underline{\mathbf{G}}$ is super-diagonal, i.e., the only non-zero elements of $\underline{\mathbf{G}}$ are in the (i,i,i) positions. This is shown pictorially in Figure 2.4. This observation is crucial in estimating the model order of PARAFAC, as we will see in Section 3.2.8

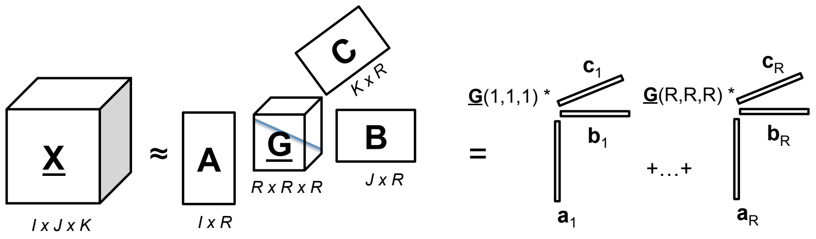


Figure 2.4: PARAFAC can be seen as restricted TUCKER, when the core is super-diagonal.

Practitioner's Guide

Strengths: Captures non-trilinear variation and compresses a tensor optimally.

Weaknesses: Non unique and hard to interpret, especially if the core tensor is dense.

When to use? Tensor compression (see 3.4.1 of Chapter 3), analyzing data where relations between latent components is expected, and estimating missing values (since it can capture variation that PARAFAC cannot). Note here that if the data have low-rank multilinear structure, PARAFAC may be better than TUCKER at compressing that data, however, TUCKER can compress well datasets that do not necessarily have low tensor rank.

2.3.3 Data Fusion & Coupled Matrix/Tensor Models

2.3.3.1 Definition

In a wide variety of applications, we have data that form a tensor and have side information or meta-data which may form matrices or other tensors. For instance, suppose we have a (user, product, date) tensor that indicates which user purchased which product and when. Usually, stores also have some meta-data on the user which can form a (user, features) matrix, as well as meta-data on the products which for a (product, features) matrix. In this case, the tensor and the two matrices are *coupled* in the "user" and "product" mode respectively, since there is a one-to-one correspondence of users in the tensor and the matrix (and accordingly for the products).

In Chemometrics, this concept was first introduced by [SWB00], and [BBM07] apply this concept to data mining. There has been significant development of such coupled model, either when matrices are coupled together [SG08, AGR+12], or when matrices and tensors are coupled (a tensor can be coupled with another matrix or even another tensor) [LSC+09, ZCZ+10, AKD11, APY12, NHTK12, YCY12, WZX14, EAC15]

One of the most popular models is the so-called Coupled Matrix-Tensor Factorization (CMTF) [AKD11] where one or more matrices are coupled with a tensor. In the case where we have one coupled matrix in the "rows" mode, and we impose a PARAFAC model on the tensor, we have:

$$\min_{\mathbf{a}_r,\mathbf{b}_r,\mathbf{c}_r,\mathbf{d}_r} \|\underline{\mathbf{X}} - \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r\|_F^2 + \|\mathbf{Y} - \sum_{r=1}^{R} \mathbf{a}_r \mathbf{d}_r^T\|_F^2 \tag{2.3}$$

The CMTF decomposition with PARAFAC on the tensor consists of matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$. Notice that the factor matrix $\mathbf{A}$ corresponding to the rows of $\underline{\mathbf{X}}$ and $\mathbf{Y}$ is the same. This ensures that the two datasets are decomposed jointly and share the same latent space, effectively *fusing* the two datasets.

Instead of a PARAFAC model, we can have CMTF that assumes a TUCKER model [EAC15]:

$$\min_{\mathbf{U}_1,\mathbf{U}_2,\mathbf{U}_3,\underline{\mathbf{G}},\mathbf{D}} \|\underline{\mathbf{X}} - \underline{\mathbf{G}} \times_3 \mathbf{U}_3 \times_2 \mathbf{U}_2 \times_1 \mathbf{U}_1\|_F^2 + \|\mathbf{Y} - \mathbf{U}_1 \mathbf{D}^T\|_F^2 \tag{2.4}$$

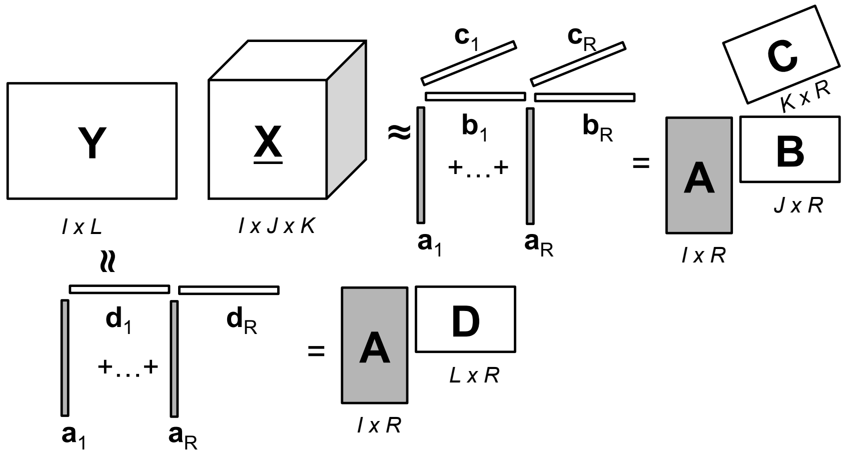


Figure 2.5: Coupled Matrix-Tensor Factorization with a PARAFAC model on the tensor. Notice that the $\mathbf{a}_r$ vectors are the same between the tensor and the matrix factorization.

An important issue when we have two or more heterogeneous pieces of data coupled with each other is the one we informally call "drowning", i.e., when one of the datasets is vastly denser or larger than the other(s) and thus dominates the approximation error. In order to alleviate such problems, we have to weight the different datasets accordingly [WCVM09].

<u>Practitioner's Guide</u>

Strengths: Jointly analyzes heterogeneous datasets that have one of their modes in common, and incorporates side-information and meta-data to the analysis.

Weaknesses: When the heterogeneous datasets are vastly different in terms of size and volume, we need to apply appropriate weighting in order to avoid one dataset "drowning" the rest.

When to use? In applications where meta-data are present and can be modeled as side-information matrices, as well as when having two (or more) heterogeneous pieces of data that refer to the same physical entities.

2.4 Conclusions

In this Chapter we presented the necessary notation used throughout this thesis, as well as the definitions for PARAFAC, TUCKER, and CMTF, which are the decompositions used in this thesis. In the next Chapter, we delve deeper into PARAFAC, TUCKER, and CMTF as well as other tensor decompositions providing algorithmic details and a broad selection of applications.

Chapter based on material to be published in [PFS].

Chapter 3

Survey of Advance Topics in Tensors and Data Fusion

Here we delve deeper into advanced topics in tensor decompositions. This chapter is not essential for understanding the basic concepts behind this thesis (and can, in fact be skipped), but serves as a reference to the interested reader who wishes to explore further and develop a working knowledge of tensor decompositions from a practitioner's point of view.

3.1 Introduction

In this Chapter we provide a survey on advanced topics in tensor decompositions, including applications and algorithmic advancements. This survey is by no means the first attempt to summarize tensor decompositions. In fact, [KB09] is probably the most concise and most cited survey that contains a very detailed overview of different tensor decompositions. Subsequently, more survey papers have been published, some of them focusing on the applications [AY09] and some on the algorithms [CZPA09, LPV11, GKT13, CMDL$^+$15]. All the aforementioned surveys are great summaries of the work in a vast research area that spans the fields of Chemometrics, Psychometrics, Signal Processing, Data Mining, and Machine Learning.

However, we feel that the present survey differs from the existing ones in the following two ways: 1) we strongly emphasize the implications of the works we summarize from a practitioner's point of view, both in terms of describing the decompositions, as well as in terms of volume and breadth of the applications that this survey contains, and 2) the field is evolving very fast and many of the advances either in applications or in scalable algorithms have been published after most of the existing surveys; however, the current state of the art is sufficiently mature for this survey to summarize and draw abstractions from, especially with respect to scalable algorithms.

In Section 3.2, picking up where we left off in Chapter 2, we continue the discussion on advanced topics for PARAFAC and TUCKER, and present a variety of other tensor decompositions that have been used in data mining. As before, we summarize each decomposition from a practitioner's point of view at the end of each sub-section. In Section 3.3, we outline the application of tensor decompositions in a broad variety of real-world applications, outlining the particular ways that tensors have been used in each case. Subsequently, in Section 3.4 we provide a concise summary of scalable methods for tensor decompositions, which have recently witnessed a significant growth, and have made tensor decompositions applicable to big data. Finally, in Section 3.5 we conclude.

3.2 Tensor Decompositions

There is a rich variety of tensor decompositions in the literature. In this Section, we provide a comprehensive overview of the most widely used decompositions in data mining, from a practitioner's point of view.

3.2.1 PARAFAC Decomposition

We saw the definition of PARAFAC in Section 2.3.1 of Chapter 2. Here we provide more details about theoretical properties of the decomposition, algorithms, and various extensions.

3.2.1.1 Theoretical properties of PARAFAC

As we saw in Chapter 2, PARAFAC is essentially unique, under very mild conditions. Formally, if $\mathbf{A}_o$ is the true latent factor matrix for the first mode, and $\mathbf{A}$ is the result of PARAFAC, then

$$\mathbf{A} = \mathbf{A}_o \mathbf{\Pi} \mathbf{\Lambda}$$

where $\mathbf{\Pi}$ is a permutation matrix that allows for switching the order of the columns of $\mathbf{A}_o$, and $\mathbf{\Lambda}$ is a diagonal matrix that scales each column of $\mathbf{A}_o$. Same holds for the rest of the factor matrices. It is beyond the scope of this survey to delve deeper into the theoretical underpinnings of PARAFAC uniqueness, however, the relevant work, put into context, is as follows: [Kru77] gave the first such uniqueness result and [SB00] extended that to tensors with four or more modes. It was known that the condition of [Kru77] is sufficient but not necessary, however [JS04] showed that it is far from necessary when one mode is full column rank. [StBDL06] used [JS04] to derive an almost-sure uniqueness result. Finally, [CO12] provided the most relaxed conditions to date, using algebraic geometry.

3.2.1.2 Algorithms

In this section we will elaborate on the Alternating Least Squares (ALS) algorithm for PARAFAC, which is probably the most widely used algorithm. We also mention in passing first-order gradient approaches. For a detailed comparison of algorithms for PARAFAC, [TB06] and [KB09] are excellent sources.

In order to solve for PARAFAC, we usually minimize the sum of squares of the difference between the tensor $\underline{\mathbf{X}}$ and the model:

$$\min_{\mathbf{a}_r,\mathbf{b}_r,\mathbf{c}_r} \|\underline{\mathbf{X}} - \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r\|_F^2$$

Notice that the above function is non-convex, however, if we fix two of the three vectors of PARAFAC, it is linear with respect to the third one. This is a key observation which we will use in order to derive the ALS algorithm in the following lines.

In addition to Equation 2.1, PARAFAC can be also written in the "slab" format, relating each "slice" of the tensor to the factor matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$. In particular, the k-th slice of an $I \times J \times K$ tensor $\underline{\mathbf{X}}$ can be written as follows:

$$\underline{\mathbf{X}}(:,:,k) = \mathbf{A} Diag(\mathbf{C}(k,:))\mathbf{B}^T$$

where $Diag(\mathbf{C}(k,:))$ is a diagonal matrix with the k-th row of $\mathbf{C}$ in the diagonal. Using Property 2.1, after vectorizing the k-th slice of the tensor into a long $IJ \times 1$ vector, we can write:

$$vec(\underline{\mathbf{X}}(:,:,k)) \approx (\mathbf{B} \odot \mathbf{A})\,\mathbf{C}(k,:)$$

and gathering all the slices together, we can write:

$$\begin{bmatrix} vec(\underline{\mathbf{X}}(:,:,1)) & vec(\underline{\mathbf{X}}(:,:,2)) & \cdots & vec(\underline{\mathbf{X}}(:,:,K)) \end{bmatrix} \approx (\mathbf{B} \odot \mathbf{A}) \begin{bmatrix} \mathbf{C}(1,:) & \mathbf{C}(2,:) & \cdots & \mathbf{C}(K,:) \end{bmatrix}$$

which finally gives:

$$\mathbf{X}_{(3)}^T \approx (\mathbf{B} \odot \mathbf{A})\,\mathbf{C}^T$$

or

$$\mathbf{X}_{(3)} \approx \mathbf{C}\,(\mathbf{B} \odot \mathbf{A})^T$$

where matrix $\mathbf{X}_{(3)}$ is the third-mode matricization of the tensor and is equal to

$$\mathbf{X}_{(3)} = \begin{bmatrix} vec(\underline{\mathbf{X}}(:,:,1)) & vec(\underline{\mathbf{X}}(:,:,2)) & \cdots & vec(\underline{\mathbf{X}}(:,:,K)) \end{bmatrix}^T.$$

By symmetry of the PARAFAC model, we can write the first and second mode slabs of the tensor in similar ways:

$$\mathbf{X}_{(1)} \approx \mathbf{A}\,(\mathbf{C} \odot \mathbf{B})^T, \quad \mathbf{X}_{(2)} \approx \mathbf{B}\,(\mathbf{C} \odot \mathbf{A})^T, \quad \mathbf{X}_{(3)} \approx \mathbf{C}\,(\mathbf{B} \odot \mathbf{A})^T$$

The above observation is very crucial, because it offers three linear equations that relate a matricized version of the tensor (which is known), to one of the factor matrices. Thus, assuming that $(\mathbf{B} \odot \mathbf{C})$ is fixed, we can solve for $\mathbf{A}$ as

$$\hat{\mathbf{A}} = \mathbf{X}_{(1)}\,(\mathbf{C} \odot \mathbf{B})^\dagger$$

which, after using a property of the Khatri-Rao product, turns into:

$$\hat{\mathbf{A}} = \mathbf{X}_{(1)}\,(\mathbf{C} \odot \mathbf{B})\left(\mathbf{C}^T\mathbf{C} * \mathbf{B}^T\mathbf{B}\right)^\dagger$$

For the general N-mode case, where we have factors $\mathbf{A}_1 \cdots \mathbf{A}_N$, the ALS solution is:

$$\hat{\mathbf{A}}_{(n)} = \mathbf{X}_{(n)} \left(\mathbf{A}_N \odot \cdots \odot \mathbf{A}_{n+1} \odot \mathbf{A}_{n-1} \odot \cdots \odot \mathbf{A}_1\right) \left(\mathbf{A}_N^T \mathbf{A}_N * \cdots * \mathbf{A}_{n+1}^T \mathbf{A}_{n+1} * \mathbf{A}_{n-1}^T \mathbf{A}_{n-1} * \cdots * \mathbf{A}^{(1)T} \mathbf{A}^{(1)}\right)^{\dagger}$$

This gives rise to the ALS algorithm for the PARAFAC decomposition, which does exactly that: At each step, it assumes that two out of the three matrices are fixed and solves for the third, and does so iteratively until a convergence criterion is met, e.g., when the approximation error stops decreasing, or when a certain number of iterations is reached. The ALS algorithm is a type of block coordinate descent algorithm and is guaranteed to decrease the approximation error monotonically. A listing of ALS is in Algorithm 3.1.

Algorithm 3.1: Alternating Least Squares for PARAFAC

```
Input: Tensor $\underline{\mathbf{X}} \in \mathbb{R}^{I \times J \times K}$ and rank $R$.
Output: PARAFAC decomposition $\mathbf{A} \in \mathbb{R}^{I \times R}, \mathbf{B} \in \mathbb{R}^{J \times R}, \mathbf{C} \in \mathbb{R}^{K \times R}$
1: Initialize $\mathbf{A}, \mathbf{B}, \mathbf{C}$ (e.g., at random)
2: while convergence criterion is not met do
3:     $\mathbf{A} \leftarrow \mathbf{X}_{(1)} (\mathbf{C} \odot \mathbf{B}) (\mathbf{C}^T\mathbf{C} * \mathbf{B}^T\mathbf{B})^{\dagger}$
4:     $\mathbf{B} \leftarrow \mathbf{X}_{(2)} (\mathbf{C} \odot \mathbf{A}) (\mathbf{C}^T\mathbf{C} * \mathbf{A}^T\mathbf{A})^{\dagger}$
5:     $\mathbf{C} \leftarrow \mathbf{X}_{(3)} (\mathbf{B} \odot \mathbf{A}) (\mathbf{B}^T\mathbf{B} * \mathbf{A}^T\mathbf{A})^{\dagger}$
6: end while
```

It is important to note that the operation

$$\mathbf{Y} = \mathbf{X}_{(1)} (\mathbf{C} \odot \mathbf{B}) \tag{3.1}$$

has a special name: *Matricized Tensor Times Khatri-Rao Product* or *MTTKRP* for short. It can be similarly defined for $\mathbf{X}_{(2)}$ and $\mathbf{X}_{(3)}$ matricizations, as per the Alternating Least Squares Algorithm. This operation is key for scalability, as we will discuss in Section 3.4.1.

In addition to the ALS algorithm, another popular approach is a first-order gradient algorithm [ADK11] where we optimize over all the variables all at once. In order to do that, [ADK11] define the gradient of the objective function, and given the gradient, they use a first-order gradient optimization solver to solve for PARAFAC.

3.2.1.3 Handling Missing Values

When dealing with real data, there are many reasons why one should expect some pieces of the data to be missing. Either because of corruption, or faulty measurements, or cases where a data point has not been observed *yet* (e.g., in Recommender Systems), there is a need to equip our algorithms with the ability to handle missing data. The traditional way for doing so is by operating only on the observed data, ignoring every entry that is missing. Mathematically, this is translated to:

$$\min_{\mathbf{a}_r,\mathbf{b}_r,\mathbf{c}_r} \|\underline{\mathbf{W}} * \left(\underline{\mathbf{X}} - \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r \right) \|_F^2$$

where

$$\underline{\mathbf{W}}(i,j,k) = \begin{cases} 1 & \text{if } (i,j,k) \text{ element is present} \\ 0 & \text{if } (i,j,k) \text{ elemeng is missing} \end{cases}$$

Intuitively, the element-wise multiplication of the difference between the data and the model with this mask tensor $\mathbf{W}$, disregards entries that are missing from the optimization process. The works of [TB05, ADKM10] show how this can be done algorithmically. Furthermore, [TB05] describes and *Expectation-Maximization* approach, where one starts by ignoring the missing values, computes a model without the missing values, uses the reconstructed tensor to estimate those missing values and iterates until the missing values converge.

3.2.1.4 Extensions

Due to its popularity, there exist a variety of extensions to the PARAFAC decomposition, inspired by real-world constraints.

Non-negative Tensor Factorization

In their seminal paper [LS99], the authors demonstrate that enforcing non-negativity constraints on the factors of a matrix factorization can lead to more interpretable and intuitive results. Conceptually, when we are dealing with data that can be expressed as a "sum-of-parts", then incorporating non-negativity constraints successfully expresses this property. In tensors, there exist algorithms that enforce non-negativity constraints on the values of $\mathbf{A}, \mathbf{B}, \mathbf{C}$ and have been shown to be more effective in terms of interpretability [SH05, CZPA09].

Sparsity on the latent factors

Another type of constraints that has been typically in exploratory mining to enhance interpretability is sparsity. When the latent factors have only a few non-zero entries, then it is more immediate and easy for an analyst to simply inspect those, ignoring the zero entries, and try to understand the result of the decomposition. In addition to interpretability, [PSB13] show that sparsity on the latent factors of PARAFAC actually leads to higher-way co-clustering, which in contrast to plain clustering, simultaneously groups rows, columns, and third mode fibers (in case of a three-mode tensor), into so-called co-clusters. In order to achieve sparsity on the latent factors, it has been shown that adding penalties on the ℓ_1 norm of the factors, i.e., the sum of their absolute values, achieves that:

$$\min_{\rho_r,\mathbf{a}_r,\mathbf{b}_r,\mathbf{c}_r} \|\underline{\mathbf{X}} - \sum_{r=1}^{R} \rho_r \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r\|_F^2 + \lambda_a \sum_{i,r} |\mathbf{a}_r(i)| + \lambda_b \sum_{j,r} |\mathbf{b}_r(j)| + \lambda_c \sum_{k,r} |\mathbf{c}_r(k)|$$

On the above decomposition, the columns of $\mathbf{A}, \mathbf{B}, \mathbf{C}$ are normalized to unit norm. Furthermore, another interesting observation in [PSB13] is that by having sparse latent factors, PARAFAC in "deflation" mode produces numerically accurate results, as the extraction of the factors is done all-at-once. Deflation refers to the process of extracting one rank-one component, subtracting it from the data and iterating for all desired components. As we will see in Section 3.4.1, this idea is one way of speeding up PARAFAC, and if we impose sparsity on the latent factors, because of the fact that they are nearly orthogonal, it works accurately.

Boolean PARAFAC *Decomposition*

Oftentimes, tensors contain binary relations (e.g., when we are recording whether two people spoke on the phone or not on particular day). In [Mie11], the authors proposes a set of Boolean tensor decompositions (PARAFAC and TUCKER), where binary tensors are decomposed using models that operate in the Boolean algebra (e.g., $1 + 1 = 1$). The advantage of this approach is that for tensors that contain binary relations, such representation is more appropriate and respects that nature of the data.

Modelling sparse count data

In a wide variety of data mining applications, we are dealing with sparse "count" data, .e.g, how many times did two people on a social network interact during a week. Traditionally, tensor algorithms seek to minimize the Frobenius norm of the difference between the data and the model, which has proven to be quite effective in defining robust and well studied algorithms. However, using the Frobenius norm assumes that the data are normally distributed. In the case of sparse count data, this assumption is not well suited and may lead to results that misrepresent the data. On the other hand, postulating a Poisson distribution for the data turns out to be a more realistic assumption, which implies the use of the KL-Divergence as an objective function. The works of [LSC+09] and [CK12] demonstrate the effectiveness of this assumption.

Incremental Decomposition

Finally, in many real-world scenarios, the tensor is not a static piece of data, but is generated dynamically and updated over time. Suppose, for instance, that the third mode of the tensor is being updated by new slices of data, e.g., new interactions on a time-evolving social network. Instead of re-computing the PARAFAC decomposition of the updated data every time a new slice arrives, [NS09] propose a method that tracks the decomposition, updating the factors given the new data, avoiding the overhead of re-running the full-algorithm for every update.

3.2.2 TUCKER Decomposition

We saw the definition of TUCKER in Section 2.3.2 of Chapter 2. Here we elaborate on algorithms and various extensions.

3.2.2.1 Algorithms

There exist two very popular algorithms for computing the TUCKER decomposition. The first one is called Higher Order Singular Value Decomposition [DLDMV00] and the main idea behind it is that for each $\mathbf{U}_i$, it computes the R_i leading singular vectors of the i-th matricization of tensor $\mathbf{X}$. Having computed all $\mathbf{U}_i$ this way, it then computes the core tensor $\underline{\mathbf{G}}$ conditioned on those matrices. In order to find the solution for $\underline{\mathbf{G}}$, we can rewrite the TUCKER model as:

$$vec\left(\underline{\mathbf{X}}\right) \approx \left(\mathbf{U}_3 \otimes \mathbf{U}_2 \otimes \mathbf{U}_1\right) vec\left(\underline{\mathbf{G}}\right)$$

The least squares solution for $vec\left(\underline{\mathbf{G}}\right)$ is

$$vec\left(\hat{\underline{\mathbf{G}}}\right) = \left(\mathbf{U}_3 \otimes \mathbf{U}_2 \otimes \mathbf{U}_1\right)^{\dagger} vec\left(\underline{\mathbf{X}}\right)$$

which due to a property of the Kronecker product becomes

$$vec\left(\hat{\underline{\mathbf{G}}}\right) = \left(\mathbf{U}_3^{\dagger} \otimes \mathbf{U}_2^{\dagger} \otimes \mathbf{U}_1^{\dagger}\right) vec\left(\underline{\mathbf{X}}\right)$$

By orthogonality of the columns of $\mathbf{U}_n$, it holds that $\mathbf{U}_n^{\dagger} = \mathbf{U}_n^T$, therefore

$$vec\left(\hat{\underline{\mathbf{G}}}\right) = \left(\mathbf{U}_3^T \otimes \mathbf{U}_2^T \otimes \mathbf{U}_1^T\right) vec\left(\underline{\mathbf{X}}\right)$$

and by folding back the vectorized tensor to its original form, we have

$$\hat{\underline{\mathbf{G}}} = \underline{\mathbf{X}} \times_3 \mathbf{U}_3^T \times_2 \mathbf{U}_2^T \times_1 \mathbf{U}_1^T$$

With the above equation, we conclude the computation of TUCKER using the HOSVD algorithm. A listing of HOSVD is shown in Algorithm 3.2. Note that HOSVD is not computing the optimal solution to the decomposition and does not compute joint subspaces of the tensor. However, it has been widely used due to its simplicity.

Algorithm 3.2: Higher Order Singular Value Decomposition (HOSVD)

```
Input: $N-$mode tensor $\underline{\mathbf{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and ranks $R_1, \cdots, R_N$.
Output: TUCKER factors $\mathbf{U}_1 \in \mathbb{R}^{I_1 \times R_1}, \cdots \mathbf{U}_N \in \mathbb{R}^{I_N \times R_N}$ and core tensor $\underline{\mathbf{G}} \in \mathbb{R}^{R_1 \times \cdots \times R_N}$
1: for $n = 1 \cdots N$ do
2:     $[\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}] \leftarrow \text{SVD}(\mathbf{X}_{(n)})$
3:     $\mathbf{U}_n \leftarrow \mathbf{U}(:, 1 : R_n)$, i.e., set $\mathbf{U}_n$ equal to the $R_n$ left singular vectors of $\mathbf{X}_{(n)}$
4: end for
5: $\underline{\mathbf{G}} \leftarrow \underline{\mathbf{X}} \times_N \mathbf{U}_N^T \times_{N-1} \mathbf{U}_{N-1}^T \cdots \times_1 \mathbf{U}_1^T$
```

The solution of HOSVD can also be used as a starting input for the second algorithm for TUCKER, an ALS method called Higher Order Orthogonal Iteration (HOOI). In order to derive HOOI, we need to compute the conditional update of $\mathbf{U}_i$ given the rest of the factor matrices. After manipulating the objective function (a detailed derivation can be found in [KB09]), it turns out that in order to solve for $\mathbf{U}_1$ we need to maximize:

$$\max_{\mathbf{U}_1} \|\mathbf{U}_1^T \underbrace{\left(\mathbf{X}_{(1)} \left(\mathbf{U}_3 \otimes \mathbf{U}_2\right)\right)}_{\mathbf{W}} \|_F^2$$

The optimal solution to the above maximization are the top R_1 singular vectors of $\mathbf{W} = \mathbf{X}_{(1)} \left(\mathbf{U}_3 \otimes \mathbf{U}_2\right)$. By symmetry of the model, in order to solve for $\mathbf{U}_2$ we take the top R_2 singular vectors of $\mathbf{X}_{(2)} \left(\mathbf{U}_3 \otimes \mathbf{U}_1\right)$, and for $\mathbf{U}_3$, we take the top R_3 singular vectors of $\mathbf{X}_{(3)} \left(\mathbf{U}_2 \otimes \mathbf{U}_1\right)$. For the N-mode case, the solution for $\mathbf{U}_n$ is by taking the top R_n singular vectors of

$$\mathbf{X}_{(n)} \left(\mathbf{U}_N \otimes \cdots \mathbf{U}_{n+1} \otimes \mathbf{U}_{n-1} \otimes \cdots \otimes \mathbf{U}_1\right).$$

Combining the above updates with the update for $\underline{\mathbf{G}}$ that we showed earlier for HOSVD, we have HOOI, outlined in Algorithm 3.3.

Algorithm 3.3: Higher Order Orthogonal Iteration (HOOI) for TUCKER

```
Input: $N-$mode tensor $\underline{\mathbf{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and ranks $R_1, \cdots, R_N$.
Output: TUCKER factors $\mathbf{U}_1 \in \mathbb{R}^{I_1 \times R_1}, \cdots \mathbf{U}_N \in \mathbb{R}^{I_N \times R_N}$ and core tensor $\underline{\mathbf{G}} \in \mathbb{R}^{R_1 \times \cdots \times R_N}$
1: Initialize $\mathbf{U}_1, \cdots \mathbf{U}_n$ (e.g., at random or using Algorithm 3.2).
2: while convergence criterion is not met do
3:     for $n = 1 \cdots N$ do
4:         $\underline{\mathbf{W}} \leftarrow \underline{\mathbf{X}} \times_N \mathbf{U}_N^T \cdots \times_{n+1} \mathbf{U}_{n+1}^T \times_{n-1} \mathbf{U}_{n+1}^T \cdots \times_1 \mathbf{U}_1^T$
5:         $[\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}] \leftarrow \text{SVD}(\mathbf{W}_{(n)})$
6:         $\mathbf{U}_n \leftarrow \mathbf{U}(:, 1 : R_n)$
7:     end for
8: end while
9: $\underline{\mathbf{G}} = \underline{\mathbf{X}} \times_N \mathbf{U}_N^T \times_{N-1} \mathbf{U}_{N-1}^T \cdots \times_1 \mathbf{U}_1^T$
```

3.2.2.2 Handling Missing Values

TUCKER can handle missing values the same way as PARAFAC, by introducing the weight tensor $\underline{\mathbf{W}}$ that masks the missing entries from the optimization. In addition to handling missing values, TUCKER has been shown to be generally more effective at estimating missing values, compared to PARAFAC [KABO10]. The reason for this behavior is the fact that TUCKER, by virtue of its core that models interactions between components, can capture variation in the data that is not strictly tri-linear.

3.2.2.3 Extensions

Boolean TUCKER *Decomposition*

As we saw in PARAFAC, [Mie11] introduce Boolean tensor decomposition using the TUCKER model as well.

Incremental Decomposition

In cases where the tensor is dynamically created, or viewed as a stream of incoming slices, [STF06] introduce an algorithm that can track the decomposition without re-computing it for every incoming slice.

Handling Time

Finally, the work of [STH$^+$08] shows how to effectively handle time as one of the modes of the tensor. Due to the fact that time is treated as any other categorical attribute in a tensor (and the decompositions work as well if we permute the time positions), oftentimes time has to be treated carefully, especially when such permutation freedom violates the assumptions of the application. In [STH$^+$08] the authors show how to model time using a wavelet decomposition, and incorporate that to a TUCKER decomposition.

3.2.3 DEDICOM and related models

3.2.3.1 Definition

In many real data applications we have relations between entities (such as people in a social network) that are inherently asymmetric. For instance, when we are recording the number of e-mails that someone sent to a person, this (sender, receiver) relation is asymmetric. In order to analyze such data, there exists a tensor decomposition called DEDICOM which is able to capture such asymmetries. The decomposition was first proposed in [Har78] and was later on used in [BHK07] In DEDICOM, we write an $I \times I \times K$ tensor (note that the first two modes have the same size), as:

$$\underline{\mathbf{X}} \approx \mathbf{A}\underline{\mathbf{D}}\mathbf{R}\underline{\mathbf{D}}\mathbf{A}^T$$

$\mathbf{A}$ is the loading or embedding matrix for the rows and columns of $\underline{\mathbf{X}}$, which correspond to the same entities. The columns of $\mathbf{A}$ correspond to latent components, and as [BHK07] states, they can be thought of as *roles* that an entity in the rows of $\underline{\mathbf{X}}$ participates in. Each slice of $\underline{\mathbf{D}}$ is an $I \times I$ diagonal matrix, giving weights to the columns of $\mathbf{A}$, and $\mathbf{R}$ captures the asymmetric relations, and according to [BHK07] it captures the aggregate trends over the third mode (which in that particular paper is time, but can be anything else). The model is shown pictorially in Figure 3.1.

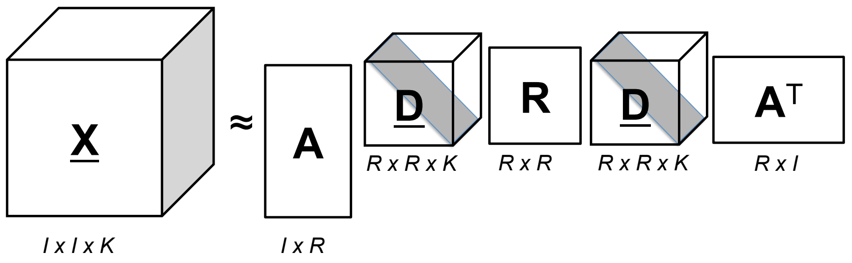


Figure 3.1: The three-mode DEDICOM decomposition.

A more recent variation of DEDICOM is called RESCAL and can be found in [NTK11]. RESCAL was especially proposed for modeling multi-relational data. The RESCAL decomposition is:

$$\underline{\mathbf{X}} \approx \mathbf{A}\underline{\mathbf{R}}\mathbf{A}^T$$

and the resemblance to DEDICOM is apparent, however, $\underline{\mathbf{R}}$ is not restricted to a particular form, and can capture more complex relations. An interesting observation here is that RESCAL is a restricted and symmetric TUCKER model, where $\mathbf{W}$ is the identity matrix (leaving the corresponding mode uncompressed), and $\mathbf{U} = \mathbf{V} = \mathbf{A}$; TUCKER models where one of the modes is uncompressed are also known as Tucker-2 [Tuc66, KB09] The decomposition is also shown in Figure 3.2.

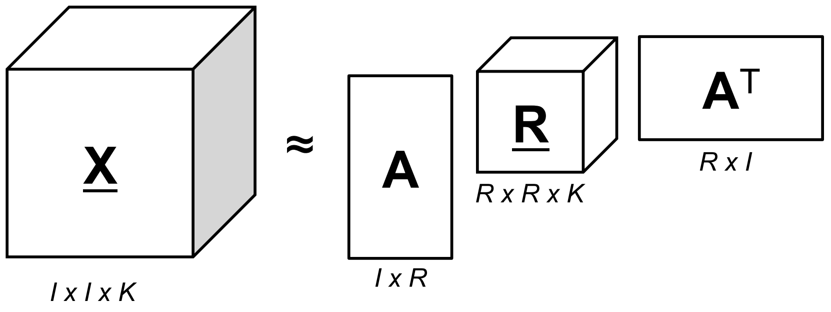


Figure 3.2: The RESCAL decomposition.

According to [NTK12], its advantage over other tensor decompositions is that it can exploit a *collective learning* effect when applied to relational data. Collective learning refers to the automatic exploitation of attribute and relationship correlations across multiple interconnections of entities and relations.

3.2.3.2 Algorithms

In this section we will provide the latest ALS algorithm for DEDICOM, which is an improvement upon the original ALS algorithm introduced in [Kie93]. Algorithm 3.4 is called ASALSAN and was proposed in [BHK07]. As all ALS algorithms, ASALSAN consists of conditional updates of $\mathbf{A}$, $\mathbf{R}$, and $\underline{\mathbf{D}}$, until convergence. We omit the derivation of the update rules which can be found in [BHK07].

Algorithm 3.4: ASALSAN algorithm for DEDICOM

```
Input: Tensor $\underline{\mathbf{X}} \in \mathbb{R}^{I \times I \times K}$ and latent dimension $R$.
Output: DEDICOM model $\mathbf{A}, \mathbf{R}, \underline{\mathbf{D}}^{I \times I \times K}$
1: Initialize $\mathbf{A}, \mathbf{R}, \underline{\mathbf{D}}$ at random (or per [BHK07]).
2: while convergence criterion is not met do
3:     $\mathbf{A} \leftarrow \left(\sum_{k=1}^{K} \left(\underline{\mathbf{X}}(:,:,k)\mathbf{A}\underline{\mathbf{D}}(:,:,k)\mathbf{R}^T\underline{\mathbf{D}}(:,:,k) + \underline{\mathbf{X}}(:,:k)^T\mathbf{A}\underline{\mathbf{D}}(:,:,k)\mathbf{R}\underline{\mathbf{D}}(:,:,k)\right)\right)$
           $\left(\sum_{k=1}^{K} (\mathbf{B}_k + \mathbf{C}_k)\right)^{-1}$, where
       $\mathbf{B}_k = \underline{\mathbf{D}}(:,:,k)\mathbf{R}\underline{\mathbf{D}}(:,:k)\left(\mathbf{A}^T\mathbf{A}\right)\underline{\mathbf{D}}(:,:,k)\mathbf{R}^T\underline{\mathbf{D}}(:,:,k)$
       $\mathbf{C}_k = \underline{\mathbf{D}}(:,:,k)\mathbf{R}^T\underline{\mathbf{D}}(:,:,k)\left(\mathbf{A}^T\mathbf{A}\right)\underline{\mathbf{D}}(:,:,k)\mathbf{R}\underline{\mathbf{D}}(:,:,k)$
4:     $vec(\mathbf{R}) \leftarrow \left(\sum_{k=1}^{K} \left(\underline{\mathbf{D}}(:,:,k)\mathbf{A}^T\mathbf{A}\underline{\mathbf{D}}(:,:,k)\right) \otimes \left(\underline{\mathbf{D}}(:,:,k)\mathbf{A}^T\mathbf{A}\underline{\mathbf{D}}(:,:,k)\right)\right)^{-1}$
           $\sum_{k=1}^{K} vec\left(\underline{\mathbf{D}}(:,:,k)\mathbf{A}^T\underline{\mathbf{X}}(:,:,k)\mathbf{A}\underline{\mathbf{D}}(:,:,k)\right)$
5:     Solve for $\underline{\mathbf{D}}$ using Newton's method:
       $\min_{\underline{\mathbf{D}}(:,:,k)} \|\underline{\mathbf{X}}(:,:,k) - \mathbf{A}\underline{\mathbf{D}}(:,:,k)\mathbf{R}\underline{\mathbf{D}}(:,:,k)\mathbf{A}^T\|_F^2$
6: end while
```

Practitioner's Guide

Strengths: Can model multi-relational data that also exhibit asymmetries. The DEDICOM model is also unique.

Weaknesses: Restricted to multi-relational data

When to use? When modeling social network data that exhibit asymmetric relations, Knowledge Bases relations or Linked Data.

3.2.4 Hierarchical Tucker Decomposition (H-Tucker)

3.2.4.1 Definition

So far, we have mostly used three-mode tensors in our examples, mostly for ease of exhibition. However, in many real world scenarios (as we will also see in Section 3.3), we have tensors of higher order. When the tensor order (i.e., the number of modes) increases, supposing that we would like to compute a TUCKER model, the number of variables we need to estimate increases exponentially to the number of modes. For example, assuming an $I \times I \times I \times I$ four-mode tensor, its (R, R, R, R) TUCKER decomposition requires the computation of R^4 values for the core tensor $\underline{\mathbf{G}}$.

This curse of dimensionality, however, can be avoided in a number of ways. The first way is the so-called Hierarchical Tucker Decomposition (H-Tucker) [HK09, Gra10, KT12, BGK13] .

The basic idea behind H-Tucker is the following: Suppose we have a binary tree of hierarchies of the modes of the tensor that can be potentially be given to us by the application (e.g., see overview of [PCVS15] in Section 3.3.7). Given this binary tree hierarchy, H-Tucker creates a set of generalized matricizations of the tensor according to each internal node of the tree. These matricizations are defined over a set of indices indicated by the particular node of the tree: for instance, given an $I \times J \times K \times L$ tensor, if node t splits the modes into two disjoint sets $\{I, J\}$ and $\{K, L\}$, then the generalized matricization $\mathbf{X}_{(t)}$ will create an $IJ \times KL$ matrix where slices of the modes that are compacted into a single mode are stacked in a systematic fashion. Details on how this matricization is done can be found in [KT12].

For each internal node of the tree it computes a "transfer" core tensor which requires the estimation of much fewer values than in the TUCKER case. The core tensor $\mathbf{B}_t$ is computed via:

$$\mathbf{U}_t = (\mathbf{U}_{t_l} \otimes \mathbf{U}_{t_r})\, \mathbf{B}_t,$$

where $\mathbf{B}_t$ is a $r_{t_l} r_{t_r} \times r_t$ matrix and $\mathbf{U}_t$ contains the r_t left singular vectors of the $\underline{\mathbf{X}}_{(t)}$ matricization. Finally, the leaf nodes contain the factor matrices that are similar to the ones that TUCKER would give.

A pictorial example of a binary tree hierarchy and its corresponding H-Tucker decomposition is shown in Figure 3.3.

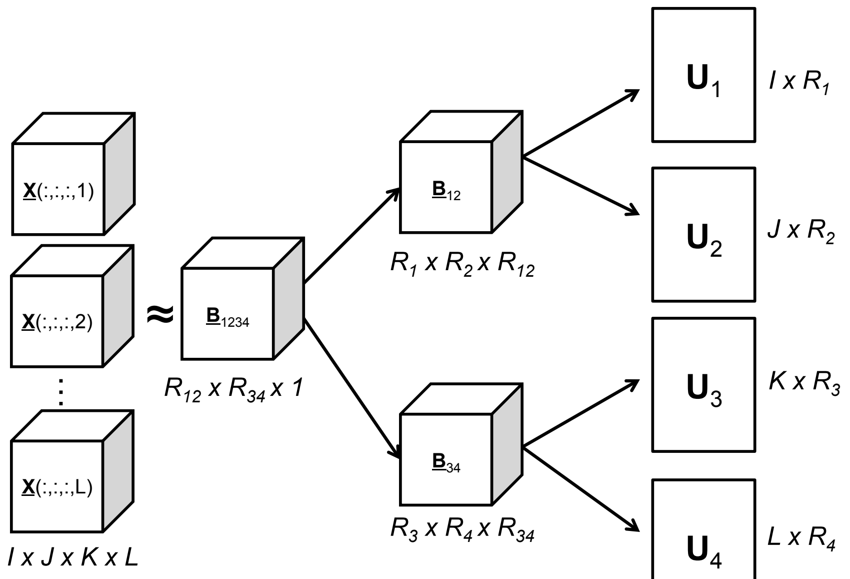


Figure 3.3: The H-Tucker decomposition.

3.2.4.2 Algorithms

The algorithm for computing H-Tucker is described in [KT12], and here we provide an outline, in Algorithm 3.5.

Algorithm 3.5: H-Tucker Algorithm

Input: $N-mode$ tensor $\underline{\mathbf{X}}$, ranks $R_1, \cdots, R_N$, and a binary tree $\mathcal{T}$ of the matricizations of $\underline{\mathbf{X}}$.
Output: H-Tucker Decomposition $\mathbf{U}_1, \cdots \mathbf{U}_N$, $\underline{\mathbf{B}}_t$ where $t \in N(\mathcal{T})$ (i.e., the non-leaf nodes of binary tree $\mathcal{T}$).

```
1: for $n = 1 \cdots N$ do
2:     $[\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}] \leftarrow \text{SVD}(\mathbf{X}_{(n)})$
3:     $\mathbf{U}_n \leftarrow \mathbf{U}(:, 1 : R_n)$, i.e., set $\mathbf{U}_n$ equal to the $R_n$ left singular vectors of $\mathbf{X}_{(n)}$
4: end for
5: Starting from the root of tree $\mathcal{T}$, select a node $t$.
6: $t_l$ is the left child, $t_r$ is the right child.
7: $\mathbf{B}_t \leftarrow \left(\mathbf{U}_{t_l}^T \otimes \mathbf{U}_{t_r}^T\right) \mathbf{U}_t$
   where $\mathbf{B}_t$ is a $r_{t_l} r_{t_r} \times r_t$ matrix, and $\mathbf{U}_t$ contains the $r_t$ left singular vectors of the $\underline{\mathbf{X}}_{(t)}$
   matricization.
8: Reshape $\mathbf{B}_t$ into a $r_{t_l} \times r_{t_r} \times r_t$ tensor.
9: Recurse on $t_l$ and $t_r$ until $t_l$ and $t_r$ are singletons.
```

Practitioner's Guide

Strengths: Approximates well high-order tensors (with number of modes much larger than three) without suffering from the curse of dimensionality.

Weaknesses: Requires a-priori knowledge of a binary tree of matricizations of the tensor.

When to use? For very high-order tensors, especially when the application at hand provides an intuitive and natural hierarchy over the modes.

3.2.5 Tensor-Train Decomposition (TT)

3.2.5.1 Definition

Along the same lines as H-Tucker, there is the Tensor-Train decomposition proposed in [Ose11], which tackles the curse of dimensionality in very high order tensors by imposing a parsimonious model. In contrast to H-Tucker, Tensor-Train does not require prior knowledge of a hierarchy of the modes. Tensor-Train decomposes a given tensor into a matrix, followed by a series of three-mode "transfer" core tensors (as in the case of the core tensors in H-Tucker), followed by a matrix. Each one of the core tensors is "connected" with its neighboring core tensor through a common reduced mode R_i. For a four-mode tensor, the Tensor-Train decomposition can be written as:

$$\underline{\mathbf{X}}(i,j,k,l) \approx \sum_{r_1,r_2,r_3} \mathbf{G}_1(i,r_1)\underline{\mathbf{G}}_2(r_1,j,r_2)\underline{\mathbf{G}}_3(r_2,k,r_3)\mathbf{G}_4(r_3,l)$$

A pictorial illustration of the four-mode Tensor-Train decomposition is shown in Figure 3.4. For the general d-mode case, we have:

$$\underline{\mathbf{X}}(i_1, \cdots, i_d) \approx \sum_{r_1, r_2 \cdots r_{d-1}} \mathbf{G}_1(i, r_1) \underline{\mathbf{G}}_2(r_1, j, r_2) \cdots \underline{\mathbf{G}}_{d-1}(r_{d-2}, d-1, r_{d-1}) \mathbf{G}_d(r_{d-1}, d)$$

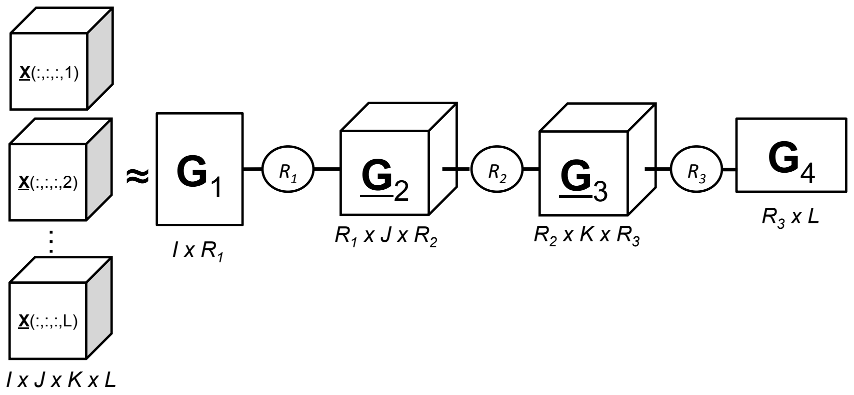


Figure 3.4: Tensor-Train Decomposition of a four-mode tensor. The circled variables indicate the reduced dimension that connects the core tensors.

3.2.5.2 Algorithms

Algorithm 3.6 outlines the computation of the Tensor-Train decomposition, as introduced in [Ose11]. Notice that the user is not required to explicitly define the reduced dimensions R_i; they are automatically calculated (line 7) by keeping the singular vectors that respect the approximation tolerance ϵ defined by the user. Of course, we can re-define the algorithm where instead of ϵ, the user explicitly provides the R_i dimensions, if this is more appropriate for a given application. Also, note that in Algorithm 3.6, for notational freedom, even if a tensor has one singleton dimension, we denote it as a tensor.

Algorithm 3.6: Tensor-Train Decomposition

```
Input: $N$-mode tensor $\underline{\mathbf{X}}$ and approximation tolerance $\epsilon$.
Output: Tensor-Train Decomposition $\mathbf{G}_1, \underline{\mathbf{G}}_2 \cdots \underline{\mathbf{G}}_{N-1} \cdots \mathbf{G}_N$.
1: Compute $\delta = \frac{\epsilon}{\sqrt{N-1}} \|\underline{\mathbf{X}}\|_F$ as the truncation parameter.
2: $\mathbf{C} \leftarrow \underline{\mathbf{X}}_{(1)}$ (select an arbitrary matricization of $\underline{\mathbf{X}}$ for initialization).
3: $r_0 = 1$
4: for $k = 1 \cdots N$ do
5:     $\mathbf{C} \leftarrow \text{reshape}\left(\mathbf{C}, [r_{k-1} I_k, \frac{\text{numel}(\mathbf{C})}{r_{k-1} I_k}]\right)$
6:     Compute a truncated SVD $[\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}]$ of $\mathbf{C}$ such that the approximation error $e \leq \delta$.
7:     $r_k \leftarrow$ rank of the SVD that achieves the above approximation.
8:     $\underline{\mathbf{G}}_k \leftarrow \text{reshape}(\mathbf{U}, [r_{k-1}, I_k, r_k])$
9:     $\mathbf{C} \leftarrow \mathbf{\Sigma} \mathbf{V}^T$
10: end for
11: $\mathbf{G}_N \leftarrow \mathbf{C}$
```

Practitioner's Guide

Strengths: Approximates well high-order tensors (with number of modes much larger than three) without suffering from the curse of dimensionality.

Weaknesses: Hard to interpret the core tensors, especially when they are dense.

When to use? For approximating very-high order tensors and there is no prior hierarchical information on the modes.

3.2.6 Data Fusion & Coupled Matrix/Tensor Models

We saw the definition of Data Fusion and Coupled Matrix-Tensor Factorization (CMTF) in Section 2.3.3 of Chapter 2. Here we present algorithms for those models, as well as discussion on advanced topics.

3.2.6.1 Algorithms

As in the case of PARAFAC, in CMTF we can also define an ALS algorithm, where we fix all but one of the matrices we are seeking to estimate. If, say, we seek to solve for $\mathbf{A}$, it turns out that we need to concatenate the two pieces of data, whose rows refer to matrix $\mathbf{A}$, i.e., the matricized tensor $\mathbf{X}_{(1)}$ and matrix $\mathbf{Y}_1$, and we can then solve for $\mathbf{A}$ as:

$$\mathbf{A} = \begin{bmatrix} \mathbf{X}_{(1)} \\ \mathbf{Y}_1 \end{bmatrix}^T \left(\begin{bmatrix} (\mathbf{B} \odot \mathbf{C}) \\ \mathbf{D} \end{bmatrix}^{\dagger} \right)^T$$

Algorithm 3.7 shows the ALS algorithm for CMTF when a tensor $\underline{\mathbf{X}}$ is coupled with three matrices $\mathbf{Y}_1, \mathbf{Y}_2, \mathbf{Y}_3$. We refer the user to [AKD11] for a gradient based approach.

Algorithm 3.7: Alternating Least Squares (ALS) Algorithm for CMTF

```
Input: $\underline{\mathbf{X}}$ of size $I \times J \times K$, matrices $\mathbf{Y}_i,\ i = 1 \cdots 3$, of size $I \times I_2$, $J \times J_2$, and $K \times K_2$
    respectively, number of factors $F$.
Output: $\mathbf{A}$ of size $I \times F$, $\mathbf{B}$ of size $J \times F$, $\mathbf{C}$ of size $K \times F$, $\mathbf{D}$ of size $I_2 \times F$, $\mathbf{G}$ of size $J_2 \times F$, $\mathbf{E}$
    of size $K_2 \times F$.
1: Initialize $\mathbf{A}, \mathbf{B}, \mathbf{C}$ using PARAFAC of $\underline{\mathbf{X}}$. Initialize $\mathbf{D}, \mathbf{G}, \mathbf{E}$ as per the model (e.g.
   $\mathbf{D} = \mathbf{Y}_1 \left(\mathbf{A}^\dagger\right)^T$).
2: while convergence criterion is not met do
3:     $\mathbf{A} = \begin{bmatrix}\mathbf{X}_{(1)} \\ \mathbf{Y}_1\end{bmatrix}^T \left(\begin{bmatrix}(\mathbf{B} \odot \mathbf{C}) \\ \mathbf{D}\end{bmatrix}^\dagger\right)^T$
4:     $\mathbf{B} = \begin{bmatrix}\mathbf{X}_{(2)} \\ \mathbf{Y}_2\end{bmatrix}^T \left(\begin{bmatrix}(\mathbf{C} \odot \mathbf{A}) \\ \mathbf{G}\end{bmatrix}^\dagger\right)^T$
5:     $\mathbf{C} = \begin{bmatrix}\mathbf{X}_{(3)} \\ \mathbf{Y}_3\end{bmatrix}^T \left(\begin{bmatrix}(\mathbf{A} \odot \mathbf{B}) \\ \mathbf{E}\end{bmatrix}^\dagger\right)^T$
6:     $\mathbf{D} = \mathbf{Y}_1 \left(\mathbf{A}^\dagger\right)^T,\ \mathbf{G} = \mathbf{Y}_2 \left(\mathbf{B}^\dagger\right)^T,\ \mathbf{E} = \mathbf{Y}_3 \left(\mathbf{C}^\dagger\right)^T$
7: end while
```

3.2.6.2 Shared and Individual Components in Coupled Decompositions

A fairly recent extension of the CMTF model has to do with shared and individual latent components between the datasets that are being jointly analyzed. For instance, in the coupled matrix-tensor case, we may assume that some of the latent components are exclusive to the tensor, some are exclusive to the matrix, and some of them are shared; this assumption gives relative freedom to the joint analysis to uncover useful structure that is jointly present in both datasets, without falsely imposing structure from one dataset to the other that perhaps does not exist. In [ALR+13, APR+14] the authors introduce such flexible coupled decompositions, In a nutshell, they introduce distinct scalar weights on the components of the tensor and the matrix and they enforce a sparsity constraint on those weights, driving some of them to zero. A zero weight for, say, a component of the tensor indicates that this component is not present in the tensor and is individual to the matrix.

3.2.7 PARAFAC2 and Decomposition of Multiset Data

3.2.7.1 Definition

Closely related to the coupled datasets that we talked about in the previous section is the concept of "multiset" data, which however merits its own discussion, because of its prevalence. A multiset dataset is a collection of matrices $\{\mathbf{X}_k\}$ for $k = 1 \cdots K$, that have one mode in common. These matrices can be seen as nearly forming a tensor, however, the non-shared mode has different dimensions, and thus it has to be handled carefully. The PARAFAC2 decomposition, introduced in [Har72b], is specifically designed for such cases. Given a multiset dataset $\{\mathbf{X}_k\}$, where the matrices share the columns but have different numbers of rows, PARAFAC2 decomposes each matrix in the multiset as:

$$\mathbf{X}_k \approx \mathbf{U}_k \mathbf{H} \mathbf{S}_k \mathbf{V}^T$$

PARAFAC2 acknowledges the differences in the rows, by introducing a set of $\mathbf{U}_k$ row factor matrices, but imposes a joint latent structure on the columns and the third mode, similar to PARAFAC. A pictorial representation of PARAFAC2 is shown in Figure 3.5.

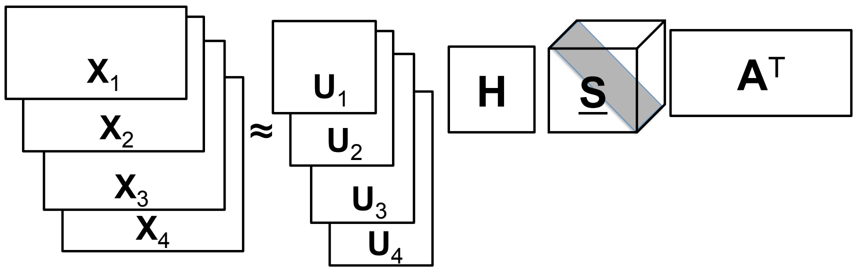


Figure 3.5: The PARAFAC2 decomposition.

3.2.7.2 Algorithms

As in many tensor decompositions, the most popular algorithm for PARAFAC2 is based on ALS [Kie93]. On Algorithm 3.8, we present an improvement upon the algorithm of [Kie93], which is proposed by [CBKA07]

Practitioner's Guide

Strengths: Can jointly analyze heterogeneous pieces of data that cannot be expressed as a tensor.

When to use? When we have a set of matrices that nearly form a tensor, but they do not match one of the modes.

3.2.8 Model Order Selection

An important issue in exploratory data mining using tensors is selecting the order of the model, in other words the rank of the decomposition in models such as PARAFAC, or the dimensions of the core tensor in Tucker-based models. This is generally a very hard problem, especially in the presence of noisy data, however there exist heuristic methods that work well in practice.

3.2.8.1 PARAFAC

For the PARAFAC decomposition, considerations about the quality of the decomposition date as early as 1984 where [Har84] outlines strategies that can reveal the quality of the decomposition. The most intuitive and popular heuristic for model order selection is the

Algorithm 3.8: ALS algorithm for PARAFAC2

```
Input: Multiset $\{\mathbf{X}_k\}$ for $k = 1 : K$ and rank $R$.
Output: PARAFAC2 Decomposition of $\{\mathbf{X}_k\}$: $\{\mathbf{U}_k\}, \mathbf{H}, \underline{\mathbf{S}}, \mathbf{V}$.
 1: Initialize:
    $\mathbf{V} \leftarrow R$ principal eigenvectors of $\sum_{k=1}^{K} \mathbf{X}_k^T \mathbf{X}_k$
    $\mathbf{H} \leftarrow \mathbf{I}$
 2: for $k = 1 \cdots K$ do
 3:     $\underline{\mathbf{S}}(:,:,k) \leftarrow \mathbf{I}$, for $k = 1 \cdots K$.
 4: end for
 5: while convergence criterion is not met do
 6:     for $k = 1 \cdots K$ do
 7:         $[\mathbf{P}_k, \mathbf{\Sigma}_k, \mathbf{Q}_k] \leftarrow$ truncated SVD of $\mathbf{H}\mathbf{S}_k\mathbf{V}^T\mathbf{X}_k^T$ at rank $R$
 8:         $\mathbf{U}_k \leftarrow \mathbf{Q}_k\mathbf{P}_k^T$
 9:     end for
10:     for $k = 1 \cdots K$ do
11:         Compute $\underline{\mathbf{Y}}(:,:,k) = \mathbf{U}_k^T\mathbf{X}_k$
12:     end for
13:     Run a single iteration of PARAFAC ALS (Algorithm 3.1) on $\underline{\mathbf{Y}}$ and compute factors $\mathbf{H}, \mathbf{V}, \hat{\mathbf{S}}$.
14:     for $k = 1 \cdots K$ do
15:         $\underline{\mathbf{S}}(:,:,k) \leftarrow Diag\left(\hat{\mathbf{S}}(k,:)\right)$
16:     end for
17: end while
```

so-called Core Consistency Diagnostic or CORCONDIA [Bro98, BK03]. CORCONDIA is based on the observation we highlighted in Section 2.3.2 and Figure 2.4, observing that PARAFAC can be written as a restricted TUCKER model with super-diagonal core. Based on that observation, CORCONDIA Given a tensor $\underline{\mathbf{X}}$ and its PARAFAC decomposition $\mathbf{A}, \mathbf{B}, \mathbf{C}$ computes a TUCKER model where $\mathbf{A}, \mathbf{B}, \mathbf{C}$ are the factors (which are known) and $\underline{\mathbf{G}}$ is the core tensor (which is unknown). The core tensor $\underline{\mathbf{G}}$ holds important information: if it is exactly or nearly super-diagonal, then $\mathbf{A}, \mathbf{B}, \mathbf{C}$ is a "good" PARAFAC decomposition (hence the chosen model order is correct), otherwise there is some problem either with the chosen rank, or with the data lacking multi-linear structure. After computing $\underline{\mathbf{G}}$, the Core Consistency diagnostic can be computed as

$$c = 100 \left(1 - \frac{\sum_{i=1}^{F} \sum_{j=1}^{F} \sum_{k=1}^{F} \left(\underline{\mathbf{G}}(i,j,k) - \underline{\mathbf{I}}(i,j,k)\right)^2}{F} \right),$$

where $\underline{\mathbf{I}}$ is a super-diagonal tensor with ones on the (i,i,i) entries. For a perfectly super-diagonal $\underline{\mathbf{G}}$ (i.e. perfect modelling), c will be 100. For rank-one models, the metric will always be 100, because the rank-one component can produces a a single scalar, which is a trivial super-diagonal core. Therefore, CORCONDIA can be used for ranks higher than 2. In [dCHR08], the authors extend the aforementioned idea, making it more robust in the presence of noise. For big tensor data, naive computation of CORCONDIA cannot scale.

However, recently [PF15] proposed a scalable algorithm for computing the diagnostic, especially for big sparse tensors.

In addition to CORCONDIA, various other methods have been proposed, ranging from Minimum Description Length (MDL) [APG$^+$14, MM15], to Bayesian [MH09, ZZC15].

3.2.8.2 TUCKER

For the TUCKER decomposition, one of the first methods for determining the model order (i.e., the size of the core tensor) is the so-called DIFFIT (DIFFerence in FIT) method [TK00]. In a nutshell, DIFFIT tries different combinations of the dimensions R_1, R_2, R_3 of the core tensor, such that $R_1 + R_2 + R_3 = s$ for various values of $s \geq 3$. For each given s, DIFFIT finds the combination of dimensions that gives the best fit-complexity (as measured by the number of fitted parameters) trade-off. Subsequently, DIFFIT compares different values of s, choosing the one $s_c = R_1 + R_2 + R_3$ that yields the best fit. DIFFIT requires the computation of multiple TUCKER decompositions, fact that may slow down the estimation of the model order. In [KK03] the authors propose a method that computes a single TUCKER decomposition and is shown to perform comparably to DIFFIT. Finally, as in the PARAFAC case, [MH09] can also estimate the model order of TUCKER employing a Bayesian framework.

3.3 Data Mining Applications

Tensors are very powerful and versatile tools, as demonstrated by the long list of their applications in data mining. In this Section, we cover a wide spectrum of such applications: Social & Collaboration Network Analysis, Web Mining & Web Search, Knowledge Bases, Information Retrieval, Topic Modeling, Brain data analysis, Recommendation Systems, Urban Computing, Healthcare & Medical Applications, Computer Networks, Speech & Image Processing, and Computer Vision. For each application, we focus on what the problem formulation is, how is a tensor modeled and which decomposition is used, and discuss the results.

3.3.1 Social & Collaboration Network Analysis

Social & collaboration network analysis can benefit from modeling data as tensors in various ways: when there exist multiple "views" of the network (e.g., who-calls-whom, who-texts-whom and so on), this can be expressed as a three-mode tensor with each frontal slice being an adjacency matrix of the network for a particular view. Furthermore, tensors have been used in modeling time-evolving networks, where each frontal slice of the tensor is a snapshot of the network for a particular point in time.

Tensor applications in social networks dates back to [AcCKY05] which is also one of the earliest tensor applications in data mining. In this particular work, the authors analyze IRC chat-room data, and create also a synthetic tensor data generator that mimics the properties of real chat-room data. The tensors are of the form (user, keyword, time) and for the purposes of demonstrating the utility of expressing the data as a tensor, the authors also create (user, keyword) and (user, time) matrices. Subsequently, they compare SVD (on the matrices), and TUCKER and PARAFAC for noiseless and noisy data; TUCKER seems to outperform the rest of the tools for noisy data, because of its flexibility to have different number of latent components per mode. The paper also touches upon two very important issues 1) temporal granularity of the data, indicating cases where the analysis breaks down due to inappropriate resolution in the "time" mode, and 2) model order selection, highlighting that using solely the approximation error as a guide to select the number of components does not necessarily imply better interpretation of the data.

In [BHK06, BHK07], the authors analyze the Enron e-mail exchange data, creating an (employee, employee, month) tensor, recording the number of emails sent from one employee to another. Given the asymmetry of the relation, the authors use DEDICOM to decompose the tensor and identify *latent roles* for the employees (with the roles discovered being labelled as "Executive", "Government Affairs", "Legal", "Pipeline") , as well as the communication pattern between roles. The strength of DEDICOM is its ability to discover asymmetric communication patters, e.g., executives tend to receive more emails than they send

In [KS08] model the DBLP collaboration network as a tensor of (author, keyword, conference) and use the TUCKER decomposition to identify groups of authors who publish on similar topics and on similar conferences.

In addition to the social interactions, [LSC+09] demonstrates that using the context behind those interactions can improve the accuracy in discovering communities. In this work the authors define a "Metagraph", a graph that encodes the context of social intercations, and subsequently propose a Metagraph Factorization, which essentially boils down to a coupled tensor and matrix factorization, involving all the various tensors and matrices that capture relations in the Metagraph. They apply their algorithm to a dataset from Digg where they record six relations in the data. They demonstrate that their proposed technique outperforms simple approaches that use the interaction frequency, as well as tensor methods that ignore the context of the interaction.

In [PFS12] and [PSB13], the authors apply the PARAFAC decomposition to the Enron e-mail exchange data, again forming an (employee, employee, month) tensor. They identify cliques of interaction, which concur with the roles identified by [BHK07]. Furthermore, [PFS12] applies the PARAFAC decomposition to a Facebook Wall posts datase that forms a (wall owner, wall poster, day) tensor and identify interesting patterns and their temporal evolution.

The authors of [AGH+14] show that orthogonal PARAFAC can be used to estimate latent factor models such as Latent Dirichlet Allocation (LDA) and Mixed Membership Block Models (MMBM), using the method of moments. In [HNHA13] the authors use orthogonal PARAFAC for recovering a Mixed Membership Block Model that detects overlapping communities in social networks, and apply it to a Facebook social network dataset (two-mode). This is a very interesting alternative perspective because the authors use the tensor decomposition to identify a model on a two-mode graph, and do not use the tensor decomposition on the graph itself.

[PAI13] introduce a PARAFAC based community detection algorithm in multi-view social networks. They analyze two different datasets: 1) "Reality-Mining", a multi-view social network with four views: phone-call, text, bluetooth (indicating proximity), and physical friendship, and 2) a network of researchers in DBLP having three views: co-author, using same keywords in publications, and citation of each others' work. The algorithm is able to identify communities with better accuracy than approaches that do not use all the views or do not exploit the higher order structure of the data.

In [APG+14], the authors use a PARAFAC based decomposition to identify communities in time-evolving graphs. They use Minimum Description Language (MDL) for identifying the number of communities that can be extracted from the data. They apply the algorithm to a dataset of phone-calls over time for a very large city, and identify various patterns of communities. Among the most frequent patterns were: 1) "Flickering Stars" which are star-like communication patterns that have a varying number of receivers over time, and 2) "Temporal Bipartite Cores" which are near bipartite cores of people communicating with each other over time.

[JCW+14] model user behavior over time and have two case studies: 1) behavior of academic researchers, and 2) behavior of "mentions" in tweets. In order to do identify user behavior over time, the authors model the problem as the decomposition of a series of tensors, each one for a particular point in time. For instance, for the tweets data we have a tensor of (source user, target user, keyword), for each time-tick. The authors propose to decompose each one of those tensors using a TUCKER-like model and impose regularization to tackle sparsity. Furthermore, in order to reduce the computational complexity, the authors propose an algorithm that carries out the decomposition incrementally, using the model from the previous time-point to estimate the new one. Theys showcase their algorithm in datasets from Microsoft Academic Search and Weibo, identifying interesting user behavior patterns over time.

Finally, in [SPBW15], the authors use dyadic events between countries in order to discover multilateral relations among them. In particular, they propose a Bayesian version of the PARAFAC decomposition, postulating a Poisson distribution on the data, which has been shown to be more effective when dealing with sparse, count data. They apply their proposed decomposition to a four mode tensor of (country-A, country-B, event-type, time-stamp) and evaluate the effectiveness of their approach by identifying well documented multilateral country relations in recent history.

3.3.2 Web Mining & Web Search

The authors of [KBK05] extend Kleinberg's HITS algorithm for authoritativeness and hubness scores of web-pages, including context information on the link. In particular, for each link, they use the anchor text as the context. This creates a three-mode tensor of (web-page, web-page, anchor text), and the PARAFAC decomposition gives the authoritativness and hubness scores (**A** are the authorities and **B** are the hubs, and **C** encodes the topic of each set of hubs and authorities). This in contrast to plain HITS (which can be seen as an SVD of the hyperlink matrix) provides more intuitive interpretation. The authors demonstrate the superiority of the approach in identifying topically coherent groups of web-pages by applying it to a custom crawl of the Web, emulating what a commercial search engine does.

In [SZL+05] the authors personalize web search by using historic click-through data of users. They construct a (user, query, page) tensor that records the clicks of a user to a particular result of a query, and they use HOSVD to take a low rank factorization of the click-through tensor. By reconstructing the tensor from its HOSVD they fill-in missing values, which can then be used as personalized result recommendations for a particular user.

In [AGP15a] the authors model the comparison between the results of different search engines using tensors. For a set of queries, they create a (query, keyword, date, search engine) tensor, and use the PARAFAC decomposition to create latent representations of search engines in the same space. They apply this tool to compare Google and Bing web search, and find that for popular queries, the two search engines have high overlap. Subsequently, in [AGP15b] the authors apply the same methodology to compare Google and Twitter based search, finding that the overlap is lower, showing the potential for social media based web search.

3.3.3 Knowledge Bases, Information Retrieval & Topic Modeling

The authors of [CBKA07] tackle the problem of Cross-language Information Retrieval, where we have parallel documents in different languages and we need to identify latent topics like in Latent Semantic Analysis (LSA)[DDF+90]; in a nutshell, LSA refers to taking a low-rank Singular Value Decomposition of a (term, document) matrix, and exploiting the low-rank structure to identify synonyms). Doing simply LSA on the concatenated set of terms for all languages and documents ignores the fact that parallel documents have same structure and ought to be clustered similarly to latent topics. To that end, the authors use the PARAFAC2 decomposition which is applied to multi-set data on a set of (term, document) matrices (terms can be different from language to language, that's why we don't have a tensor). The authors demonstrate the superiority of their approach by applying it on a set of translations of the Bible and achieving better performance than traditional LSA.

In [KPHF12] and [PFS12], the authors apply the PARAFAC decomposition to data coming from the Read The Web [RTW16] project. In particular, the data are in form (noun-phrase, noun-phrase, context-phrase). Using the PARAFAC decomposition, the authors identify latent topics that are semantically and contextually coherent, coming from each one of the rank-one components of the decomposition. Furthermore, the factor matrices of the PARAFAC decomposition define latent embeddings of the noun-phrases to a concept space; the authors find similar noun-phrases in that concept space, which are essentially "contextual synonyms", e.g., noun-phrases that can be used in the same semantics/context.

[JPKF15] apply the TUCKER decomposition to a particular snapshot of the Freebase knowledge base that contains entities and relations about music. The authors use the factor matrices of TUCKER as latent embeddings (as in the case of the PARAFAC factor matrices) and identify semantically coherent latent concepts entities and relations (e.g., "Classic Music" and"Pop/Rock Music".

In [NTK12] the authors model semantic data of type (entity1, entity2, relation) as a three-mode tensor. They analyze it using the RESCAL model, whose advantage is that it captures attribute and relationship correlations across multiple interconnections of entities and relations. They apply it to the YAGO[FGG07] Knowledge Base and show better performance in 1) predicting unknown triples, 2) collective learning (defined as "automatic exploitation of attribute and relationship correlations across multiple interconnections of entities and relations"), and 3) learning taxonomies on the knowledge base.

A higher-order extension of LSA can be found in [CYM13], where the authors integrate multiple relations between words from different information sources. They essentially use a TUCKER decomposition and they absorb the $\mathbf{W}$ factor matrix of figure 2.3 by multiplying it with the core tensor $\underline{\mathbf{G}}$ creating a different core tensor $\underline{\mathbf{S}}$, showing the analogy of this TUCKER representation and the SVD (where now they have two factor matrices and one core tensor). The authors demonstrate the performance of their proposed LSA extension by identifying antonyms and is-a relations more accurately than the state of the art.

Following the examples of [NTK12], the authors of [CYYM14] use a RESCAL-inspired decomposition to model and analyze knowledge base data. The new addition to the model are type constraints: each entity of a knowledge base has a known type (e.g., "person") , therefore, these type constraints are explicitly included in the model by excluding triples of entity-relation-entity with incompatible types from the optimization. This both saves computation and produces a more accurate result.

Finally, in a similar spirit to [HNHA13, AGH$^+$14], in [HMA$^+$14] the authors use the PARAFAC decomposition to compute the Latent Dirichlet Allocation (LDA) [BNJ03] via the method of moments. In addition to showing how LDA is computed using PARAFAC, the authors provide a distributed algorithm on the distributed platform REEF.

3.3.4 Brain data analysis

[AABB$^+$07] analyze Electroencephalogram (EEG) data from patients with epilepsy, in order to localize the origin of the seizure. To that end, they model the EEG data using a three-mode tensor (time-samples, scales, electrodes) tensor (after pre-processing the EEG measurements via a wavelet transformation). In order to analyze the EEG tensor, they use the PARAFAC decomposition: when they identify a potential seizure (which has signatures on the time and frequency domains), they use the factor vector of the third mode (the "electrodes" mode) to localize that activity. In some cases, the data contain artifacts that may shadow the seizures from the PARAFAC decomposition (such as activity caused by the movement of the eyes), and therefore have to be removed. In order to remove those artifacts, the authors use the TUCKER decomposition which can capture the subspace variation for each mode better than PARAFAC (due to its increased degrees of freedom which in turn make it harder to interpret). They identify the artifacts in the TUCKER components and they use them to remove the artifacts from the data.

When dealing with brain measurements such as EEG or Functional Magnetic Resonance Imaging (fMRI), usually researchers average data from multiple trials EEG measures different electrodes on the brain and fMRI measures 3-D pixels (or voxels in the literature). If we have access to the measurement data from different trials over time, instead of averaging, we can model the measurements as a three-mode tensor (voxel, time, trial). In brain measurements, it is not unusual for time shifts to happen, due to physical phenomena in the brain. If we analyze the tensor without accounting for those shifts, the solution may be degenerate and therefore not very useful to interpret. To solve that, the authors of [MHA$^+$08], following up on [HHL03], propose a modification to the PARAFAC model, the shift invariant CP (Shift-CP) which takes the time shift into account. The model is:

$$\underline{\mathbf{X}}(i,j,k) = \sum_{r=1}^{R} \mathbf{A}(i,r)\mathbf{B}(j-\boldsymbol{\tau}(k),r)\mathbf{C}(k,r)$$

where $\boldsymbol{\tau}$ is a vector of time-shifts which is also learned during the optimization. In [MHA$^+$08] the authors show that this can recover the latent components of fMRI and EEG data more accurately, avoiding degenerate solutions.

A generalization of the above Shift-CP model is given in [MHM11] where the authors propose the Convolutional CP (Conv-CP) model. The idea behind that model is that, unlike Shift-CP which allows for a single time delay per trial, Conv-CP can accommodate an arbitrary number of such delays per trial, within the length of the convolutive filter used (T). The Conv-CP decomposition is:

$$\underline{\mathbf{X}}(i,j,k) = \sum_{r=1}^{R} \sum_{\tau=1}^{T} \mathbf{A}(i,r)\mathbf{B}(j-\tau,r)\underline{\mathbf{C}}(k,r,\tau).$$

In this case, $\underline{\mathbf{C}}$ which serves as the convolutive filter for each trial is a tensor. The authors show that for both simulated and real data, Conv-CP outperforms Shift-CP since it is a more general and flexible model.

In [DGCW13] the authors have fMRI measurements over time and wish to estimate regions of the brain and the connections between them. They model the data as a tensor (x-coordinates, y-coordinates, time), and assume that using a PARAFAC decomposition, each rank-one tensor gives a particular region of the brain (the first two modes in space and the third in time). In order to guide the decomposition to find the right structure in the brain, they use linear constraints for the spatial factors of the decomposition according to groups of voxels in the brain that are known to behave in a coherent manner. Applying those constraints, the authors are able to detect nodes and the network between those nodes with higher accuracy.

[HKP+14] extend supervised learning models such as Support Vector Machines to operate on tensors as opposed to vectors or points. They leverage the fact that data such as fMRI brain scans have an inherent tensor structure that should be exploited, and propose a Kernel SVM that uses the PARAFAC decomposition of the data points as a compact representation that preserves the structure. They apply their approach in classifying fMRI brain scans from patients with Alzheimer's disease, ADHD, and brain damage due to HIV, and demonstrate the effectiveness of exploiting the higher-order structure of the data rather than ignoring it.

Finally, in [PMS+14] the authors seek to identify coherent regions of the brain, among different individuals, that have exhibit activity for groups of semantically similar stimuli. They use fMRI data from nine different people, when shown 60 different simple English nouns (e.g., "dog", "airplane", "hammer") forming a (noun, voxel, person) tensor. They also use semantic feautres for those same nouns, represented by a (noun, feature) matrix, and they use Coupled Matrix-Tensor Factorization to identify latent clusters of nouns, voxels, people, and noun features. In an unsupervised way they identify sets of semantically similar nouns that activate coherent regions of the brain such as the pre-motor cortex which is activated when holding small things or picking things up.

3.3.5 Recommendation Systems

One of the first attempts to apply tensors to collaborative filtering and recommendation systems is [XCH+10]. The authors propose to extend Bayesian Probabilistic Matrix Factorization (BPMF) [SM08] which is widely use in Recommendation Systems in the case where we have temporal information. They propose a Bayesian Probabilistic Tensor Factorization (BPTF) which is based on the PARAFAC model. In experimental evaluation, they show that BPTF outperforms BPMF, demonstrating that using temporal information and exploiting the higher order structure it induces on the data proves beneficial for recommendation.

In a similar spirit as above and around the same time, the authors of [KABO10] propose to use context (such as time) in traditional user-item recommendation scenarios by modeling the data as a tensor. The difference is that they use HOSVD to take a low rank decomposition of the data (only on the observed values) and use the reconstructed values for the missing values. The method beats Matrix Factorization techniques as well as other Context Aware Techniques that may (partially) ignore the higher order structure of the data that the tensor decomposition exploits.

[RST10] propose the Pairwise Interaction Tensor Factorization (PITF) model, as a method for tag recommendation on the web. The scenario is as follows: users tag items (webpages, pictures, products etc) over time, and we would like to recommend new items they would like to tag. The proposed PITF model is a special case of both TUCKER and PARAFAC. It explicitly models the pairwise interactions between users , tags, and items. In order to follow the original paper's notation, we rename the $\mathbf{A}, \mathbf{B}, \mathbf{C}$ PARAFAC factor matrices to $\mathbf{U}, \mathbf{I}, \mathbf{T}$ corresponding to the users, items, and tags respectively. To model the pairwise interactions, we divide those factor vectors as $\mathbf{U} = \begin{bmatrix}\mathbf{U}^{(I)} & \mathbf{U}^{(T)}\end{bmatrix}$, where each one of the column blocks of $\mathbf{U}$ interacts with the corresponding columns of $\mathbf{I}$ and $\mathbf{T}$. Then, the PITF model is defined as

$$\underline{\mathbf{X}}(u,i,t) = \sum_{r=1}^{R} \mathbf{U}^{(T)}(u,r)\mathbf{T}^{(U)}(t,r) + \sum_{r=1}^{R} \mathbf{I}^{(T)}(i,r)\mathbf{T}^{(I)}(t,r) + \sum_{r=1}^{R} \mathbf{U}^{(I)}(u,r)\mathbf{I}^{(U)}(i,r)$$

The authors evaluate the proposed model in comparison to PARAFAC and TUCKER, where it obtains higher prediction accuracy faster. Furthermore, PITF won the ECML/ PKDD Discovery Challenge 2009.

In [Ren10], the author introduces Factorization Machines, a generalization of Support Vector Machines which parametrizes the data internally using a factorization model, instead of using the raw data. The Factorization Machine can have degree 2 or higher. In the case of degree 2, the internal factorization is a bilinear matrix factorization, whereas for higher degrees, it uses a PARAFAC model. Factorization Machines combine the benefits of SVMs and Factorization Models, especially in scenarios of highly sparse data (such as in Collaborative Filtering) where simple SVM fails. In experiments, the author shows that Factorization Machines achieve same recommendation quality as the PITF method described above.

The authors of [PZZW10] work on a similar social-tagging scenario to [RST10] where they design a tag recommendation system on a (user, item, tag) tensor. In this paper, the authors propose an HOSVD based dimensionality reduction. They compare their proposed recommendation scheme against methods that do not fully exploit the inherent higher order structure and demonstrate better performance.

In [ZCZ+10, ZZXY12] the authors attack the problem of location-based activity recommendation to users, using side information for users, activities, and locations. In particular, the data in the problem include a tensor (user, location, activity), a (user, user) matrix of user similarities in a social network, a (location, feature), a (user, location) matrix that encodes information of a user being present at a location without a specified activity, and an (activity, activity) matrix containing the activities similarities. The authors propose to jointly decompose these datasets using a flavor of Coupled Matrix-Tensor Factorization, imposing additional proximity constraints of the involved entities in the latent space. They show that by integrating all these pieces of auxiliary information, they outperform recommendation systems that include a part of these additional sources, or none of them.

Finally, in [PK15] the authors work on image and tag recommendation on Flickr. They model the data as a multi-set where we have three matrices: (image, feature), (image, tag keyword), and (image, user). Since the data do not strictly form a tensor, the authors apply PARAFAC2 to this dataset and demonstrate its ability to obtain a joint low rank representation of this dataset which can provide high quality image and tag recommendations.

3.3.6 Urban Computing

Urban Computing refers to a class of applications that study human activity and mobility within a city, with ultimate goal to improve the livability of an urban environment.

[MDMT11] use historical data from a metropolitan area in order to forecast areas with potential future crime activity. They model the data as a four mode tensor where the first three modes are (longitude, latitude, time) and the fourth mode consists of features such as residential burglary information, social events, and offender data. They use a form of TUCKER decomposition to obtain a low-rank representation of that tensor, and they use that representation for linear discriminant analysis, to predict future crime activity.

In [WZX14] the problem the authors solve is the one of estimating the travel time of a particular trajectory in a city road network. In order to do so, they use real GPS data of travel times by a set of drivers. The issue that arises with these types of data is the high degree of sparsity, since many road segments may have not been traversed at all (or sometimes ever), thus, trying to estimate travel times for all road segments this way may result in inaccurate measurements. To alleviate this data sparsity, the authors propose to use historical data for those GPS trajectories, as well as side information about time-slots and road segments, to fill-in missing travel time values. In the heart of their proposed method lies a Coupled Matrix Tensor Factorization: they form a (road segment, driver, time-slot) tensor, with each entry containing the travel time that a driver did on a particular road segment during a particular time-slot, a (time-slot, time-slot) matrix capturing the similarities between time-slots, and a (road segment, geographical features) matrix, providing additional information for the road segments. Interestingly, the proposed Coupled Matrix Tensor Factorization here imposes a TUCKER model on the tensor part (as opposed to the PARAFAC model shown in Fig. 2.5) and this is because the primary purpose of this low rank decomposition is to complete missing values, where TUCKER can capture more non-trilinear variation in the data. Using this tensor decomposition scheme to enrich the data, the authors show that their method outperforms state of the art baselines.

[ZLW$^+$14] analyze noise complaint data from New York City, in order to identify the major sources of noise pollution in the city, for different times of the day and the week. The issue with the human generated complaints is that they result in very sparse data where some regions are overrepresented and some are underrepresented or not present at all. In a similar spirit as [WZX14], in order to overcome the data sparsity problem, the authors form a tensor out of the noise complaint data with modes (region, noise category, time-slot) and couple it with matrices with additional information for regions, noise categories and time slots. Subsequently, they decompose it using a Coupled Matrix Tensor Factorization with a TUCKER model on the tensor, and complete missing values of noise information in areas and time-slots with few or no complaints at all. The resulting dataset is an accurate representation of New York City's noise signature.

Finally, the work of [ZYW$^+$15] addresses the problem of exploring, analyzing, and estimating drivers' refueling behavior in an urban setting for better planning (e.g., placement of gas stations), and recommendation of nearby gas stations with minimal wait time. As before, the problem with the existing data created by volunteer drivers is sparsity, and the authors tackle it through low rank factorization and reconstruction of a (gas station, hour, day) tensor. In particular, they propose a flavor of HOSVD decomposition, which also incorporates context features including features of the gas station and the weather. The proposed HOSVD extension is

$$\underline{\mathbf{X}}(i,j,k) = \sum_{r_1,r_2,r_3} \mathbf{G}(i,r_1)\mathbf{H}(j,r_2)\mathbf{D}(k,r_3)\underline{\mathbf{S}}(r_1,r_2,r_3) + \sum_{l=1}^{L} \mathbf{B}(l,c_l)$$

where $\mathbf{B}(l,c_l)$ captures the effect of feature l under condition c_l in the reconstructed tensor.

3.3.7 Healthcare & Medical Applications

In [HGS14], the authors use a tensor-based technique to automatically derive phenotype candidates from Electronic Health Records. Intuitively, the problem is to automatically identify groups of patients who have similar diagnoses and have undergone similar procedures. In order to do that, they propose a tensor decomposition based on the PARAFAC model, where each phenotype candidate is a rank-one component of the PARAFAC decomposition of a (patient, diagnosis, procedure) tensor. Interestingly, the proposed decomposition, MARBLE, has a twist from the traditional PARAFAC model. In addition to the sum of rank one tensors, the proposed model also contains a rank-one "bias" tensor, as shown in the equation below:

$$\underline{\mathbf{X}} = \left(\sum_{r=1}^{R} \lambda_r \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r\right) + \left(\mathbf{u}^{(1)} \circ \mathbf{u}^{(2)} \circ \mathbf{u}^{(3)}\right).$$

In the above model, the $\mathbf{a}_r, \mathbf{b}_r, \mathbf{c}_r$ factors are constrained to be sparse, and the bias factor vectors are normally dense. In that sense, the above model can be seen as an $(R+1)$ rank PARAFAC model, where some of the columns of the factor matrices are constrained. The role of the rank-one bias tensor is to capture global and constant variation in the data that is not specific to a particular phenotype. Having this rank-one bias tensor proves instrumental in identifying good local structure in the data. In addition to the proposed model, [HGS14] also proposes to fit the above model under a KL-divergence loss and imposing non-negativity constraints, resulting in sparse and non-negative factors that are easier to interpret.

The work of [PCVS15] is the first data mining application of the Hierarchical Tucker (H-Tucker) model of Section 3.2.4. In particular, the authors propose a sparse version of H-Tucker (which is presented in more detail in Section 3.4.3) and apply it to a disease phenotyping problem, much like the one addressed by [HGS14]. The difference is that here, the authors use co-occurrence data of patients exhibiting the same disease, within a large database of medical records. The need for using H-Tucker, which handles tensors of very high order more gently than TUCKER or PARAFAC, is because the number of disease types for which the authors compute co-occurrences for is as high as 18 (which is the top-level number of disease types as defined by the International Classification of Diseases hierarchy). Thus, the tensor they operate on is as high-order as 18-mode. Using H-Tucker and interpreting the factor matrices on the leaves, the authors are able to identify meaningful phenotypes for diseases, concurring with medical experts.

3.3.8 Computer Networks

The authors of [MGF11] use the PARAFAC decomposition to analyze network traffic data from Lawrence Berkeley National Labs (LBNL) forming a tensor of (source IP, destination IP, port #, time-stamp) where each entry indicates a connection between the two IP addresses, on a given port for a given time-tick. Using the factor matrix corresponding to the "time-stamp" mode, the authors propose a spike detection algorithm on the temporal profile of the latent components of the decomposition, identifying anomalous connections in the data.

Subsequently, [PFS12] analyze the same LBNL dataset as in [MGF11], identifying components of normal activity, as well as anomalous components, such as one that indicates a port scanning network attack (where the attacker serially probes a wide range of ports in a machine, aiming to discover potential security holes).

Finally, in [MWP+14] the authors analyze two types of network data: 1) Honeynet Data of (source IP, destination IP, time-stamp) and 2) Intrusion Detection System (IDS) logs of (event type, timestamp, target IP). They apply the PARAFAC decomposition and using clustering methods on the factor matrices, for different temporal resolutions, they are able to identify anomalous events, outperforming state of the art IDS systems.

3.3.9 Speech and Image Processing & Computer Vision

The authors of [NMSP10] use the PARAFAC decomposition to conduct Blind Source Separation (BSS). BSS is the problem of estimating a set of signals that are mixed by an unknown channel (hence "blind"), using solely the information on the receiver's end as measured by a set of sensors. This problem arises in scenarios such as speech separation where each signal refers to an individual's speech, and sensors refer to microphones. The authors show that using PARAFAC can outperform other baselines, and furthermore, PARAFAC's uniqueness properties can give guarantees for the harder, under-determined version of the problem where we have more speakers (signals) than microphones (sensors).

In [LMWY13] the authors propose a tensor-based method for completing missing values in series of images. In order to do so, they define the trace norm for the tensor case, and extend matrix completion algorithms that use the matrix trace norm. The authors compare their proposed method against TUCKER, PARAFAC, and SVD each tensor slice separately, and they demonstrate that their proposed method achieves superior performance.

Pioneering the use of tensors in computer vision, [VT02] introduce TensorFaces, a methodology that uses tensors to analyze collections face images into principal components across many different modes of a picture (e.g., different pose, different illumination, or different facial expression). 28 male subjects photographed in 5 poses of 3 illuminations, and 3 expressions and 7943 pixels per image. They apply HOSVD to this ensemble of images and identify the principal variation of every picture with respect to all the modes of the data.

Finally, in [TSL+08] the authors propose a Bayesian version of HOSVD and use it in order to tackle model 3-D face data. In particular, using their proposed decomposition in ensembles of 3-D face images, they compute a parsimonious representation of the 3-F faces in the form of core tensor which captures latent characteristics of the 3-D faces, which is sufficient to later reconstruct any particular face with a given expression.

3.4 Scaling Up Tensor Decompositions

With the advent of big data, tensor decompositions have faced a set of challenges that needed to be addressed before tensors can be used for truly big data applications. In the recent years, this particular sub-field of designing scalable tensor decompositions has witnessed a remarkable growth and is currently at a sufficiently mature stage, where tensor decompositions can be deployed in the big data scale. In this section, we present such advances, mostly in chronological order, and draw high level abstractions, summarizing the key ideas and insights behind each method.

At a high level, the algorithms that we survey here can be categorized as using one or more of the following strategies in order to achieve scalability and speed:

- **Compression**: Coming up with a compressed version of the tensor and decomposing it instead of the full data is one of the earliest approaches, which has also been a recurring theme in recent works.
- **Exploiting Sparsity**: In many applications, the tensor is highly sparse (i.e., many of the values are 0), because of the very nature of the application (e.g., in Facebook, people are usually friends with a few hundreds of people, out of the 2 billion people who use Facebook, therefore such a graph would have very few connections). Exploiting this sparsity in various ways, either by re-designing the algorithm so that it carries out sparse matrix multiplications, or used in conjunction with sampling (which is the next major category), has been a very popular way of achieving scalability and efficiency.
- **Sampling**: Either via drawing a sample of a few entries of the tensor, or through extracting sampled sub-tensors from the data, sampling has been instrumental in creating approximate algorithms which achieve good accuracy (often comparable to exact algorithms), while being much more lightweight in terms of computation.
- **Parallel/Distributed Computation**: In conjunction with the rest of the techniques, many recent works have taken advantage of parallelization of the algorithms, or the use of existing high-end distributed computation environments (e.g., Hadoop/MapReduce), and thus achieve scalability.

In the next few lines, we will summarize the advances for the PARAFAC, TUCKER, and H-Tucker decompositions, for which there exist scalable algorithms. Furthermore, Tables 3.1 and 3.2 classify the algorithms we summarize with respect to the aforementioned strategies.

3.4.1 PARAFAC

Perhaps the earliest method that, by using compression, speeds up the PARAFAC decomposition is the one proposed in [BA98]. The authors use the observation that for a given PARAFAC model of a three-mode tensor, $[\mathbf{A}, \mathbf{B}, \mathbf{C}]$, the factor matrices span the subspaces of that tensor; if we call $[\mathbf{U}_1, \mathbf{U}_2, \mathbf{U}_3]$ the bases for those subspaces, we can then write $\mathbf{A} = \mathbf{U}_1\tilde{\mathbf{A}}$, $\mathbf{B} = \mathbf{U}_2\tilde{\mathbf{B}}$, and $\mathbf{C} = \mathbf{U}_2\tilde{\mathbf{C}}$, where $\tilde{\mathbf{A}}, \tilde{\mathbf{B}}, \tilde{\mathbf{C}}$ are much smaller factor matrices. This gives rise to the algorithm which first computes a TUCKER decomposition on the original tensor $\underline{\mathbf{X}}$, obtaining core $\underline{\mathbf{G}}$ and subspace basis matrices $[\mathbf{U}_1, \mathbf{U}_2, \mathbf{U}_3]$. The core is a compressed version of $\underline{\mathbf{X}}$. The algorithm then computes a PARAFAC decomposition on $\underline{\mathbf{G}}$, obtaining the compressed factors $\tilde{\mathbf{A}}, \tilde{\mathbf{B}}, \tilde{\mathbf{C}}$, and then using the TUCKER matrices, it projects those factors back to the original subspace of the tensor. The above works because it can be shown that, this is a CANDELINC [CPK80] model, essentially a PARAFAC model with linear constraints on the factors, and it can proved that matrices $\tilde{\mathbf{A}}, \tilde{\mathbf{B}}, \tilde{\mathbf{C}}$ preserve the variation of the data. Subsequently, [CFC15] based on the same principle, derive a TUCKER compression based algorithm to speed up PARAFAC with non-negativity constraints.

Subsequently, [KBK05] define a greedy algorithm for PARAFAC which is based on the computation of multiple rank-one components. In order to do so, they define the below recursive equations:

$$\begin{aligned}
\mathbf{x}^{(t+1)} &= \underline{\mathbf{X}} \times_2 \mathbf{y}^{(t)} \times_3 \mathbf{z}^{(t)} \\
\mathbf{y}^{(t+1)} &= \underline{\mathbf{X}} \times_1 \mathbf{x}^{(t+1)} \times_3 \mathbf{z}^{(t)} \\
\mathbf{z}^{(t+1)} &= \underline{\mathbf{X}} \times_1 \mathbf{x}^{(t+1)} \times_2 \mathbf{y}^{(t)}
\end{aligned}$$

The above recursion is called higher-order power method and converges to vectors $\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*$ which can be shown to be equal to the rank-one PARAFAC decomposition of $\underline{\mathbf{X}}$, assuming that $\underline{\mathbf{X}}$ is rank-one and noise-free. The power method for tensors dates back to [KR02], wherein the authors propose that method for the symmetric case of $\mathbf{x} = \mathbf{y} = \mathbf{z}$. The power method is very advantageous for sparse tensors, since its complexity is in the order of number of non-zero entries in $\underline{\mathbf{X}}$.

This gives us a way of computing a single rank-one component, therefore, in [KBK05] the authors adopt a "deflation" technique where they compute a rank-one component at every iteration, remove it from the data, and continue iterating until all R components are extracted. This approach is not optimal, since PARAFAC factors are not orthogonal, however, it has been shown that for sparse factors, deflation numerically converges to the same factors as if they were calculated all at the same time [PSB13].

Later on, [APG+14] used the same principle of rank-one component calculation via higher-order power method, and deflation, with the addition of a Minimum Description Length (MDL) cost function, which dictates the termination of the deflation (and chooses the number of components) automatically.

A significant portion of the literature is devoted to scaling up the ALS algorithm for PARAFAC. In Section 3.2.1.2 and Equation 3.1 we define the operation MTTKRP as

$$\mathbf{Y} = \mathbf{X}_{(1)} \left(\mathbf{C} \odot \mathbf{B} \right).$$

When the dimensions $I \times J \times K$ of $\underline{\mathbf{X}}$ are big, materializing the Khatri-Rao product $(\mathbf{C} \odot \mathbf{B})$ and carrying out MTTKRP can be prohibitive. This turns out to be a scalability problem of the Alternating Least Squares algorithm, which we will henceforth refer to as "Intermediate Data Blowup" or"Intermediate Data Explosion" (a term first coined by [KS08] and later by [KPHF12]), since $(\mathbf{C} \odot \mathbf{B})$ is an interim piece of data which is a by-product of the algorithm and not the output, which, however, requires immense storage and computational resources.

The "Intermediate Data Explosion" issue was first identified and addressed by [BK07], wherein the authors define a suite of operations for sparse tensors with specific focus in Matlab. This later became the widely used Tensor Toolbox for Matlab [BK+15], which specializes in the manipulation of sparse tensors. Sparsity is key in order to achieve scalability, since it allows for the manipulation of the non-zero values of $\underline{\mathbf{X}}$ in operations such as MTTKRP, which leads to far more efficient algorithms. In particular, in [BK07], the authors show how instead of computing the Khatri-Rao product, this expensive operation can be carried out for every column of $\mathbf{C}$ and $\mathbf{B}$ independently. In this column-wise view of the operation, the computation simply reduces to n-mode products of $\underline{\mathbf{X}}$ and the columns of $\mathbf{C}$ and $\mathbf{B}$:

$$\mathbf{Y}(:, r) = \underline{\mathbf{X}} \times_3 \mathbf{C}(:, r) \times_2 \mathbf{B}(:, r)$$

for $r = 1 \cdots R$. The n-mode products of a sparse tensor $\underline{\mathbf{X}}$ with the columns of $\mathbf{C}$ and $\mathbf{B}$ can be carried out efficiently, without ever computing $(\mathbf{C} \odot \mathbf{B})$, which makes the proposed algorithm in [BK07] very efficient for sparse tensors.

Subsequently, [KPHF12] propose a mathematically equivalent way of avoiding the Intermediate Data Explosion which is more appropriate for distributed computation, proposing the first PARAFAC ALS algorithm for the Map/Reduce framework. Instead of using n-mode products, the authors decouple the Khatri-Rao product between $\mathbf{C}$ and $\mathbf{B}$, separately compute products of each column $\mathbf{C}(:, r)$ and $\mathbf{B}(:, r)$ with $\mathbf{X}_{(1)}$, and then combine the results using Hadamard (element-wise) products. The reason why [KPHF12] follows this way, is because it is more amenable to the Map/Reduce programming model which they use in order to distribute the computation of the ALS algorithm among a cluster of machines. In addition to implementing an efficient MTTKRP operation, [KPHF12] also introduces Map/Reduce optimizations that further enable scalability for tensors that do not fit in memory. Recently, they follow up with [JPKF15] which is more efficient than [KPHF12] in the Map/Reduce framework.

More recently, [CV14] provide another efficient algorithm for computing the MTTKRP operation for sparse tensors. In particular, the authors show that the MTTKRP of Equation 3.1 can be equivalently computed in a column-wise manner as follows:

$$\mathbf{M} = \mathbf{X}_{(2)}^T \mathbf{B}(:, r)$$
$$\mathbf{Y}(:, r) = \mathbf{M}\mathbf{C}(:, r)$$

All of the above operations are very efficient assuming that $\underline{\mathbf{X}}$ is sparse. Furthermore, the authors implement the ALS algorithm using this version of MTTKRP in a shared-memory parallel architecture, efficiently scaling up the decomposition.

In [RSSK14], the authors provide a memory-efficient algorithm for MTTKRP that exploits sparsity and has memory requirements in the order of the non-zero entries of the tensor.

The latest work that is improving upon the MTTKRP operation is [SRSK15] wherein the authors, instead of computing the result of MTTKRP in a column-wise fashion, as the rest of the existing methods, they compute each row $\mathbf{Y}(i, :)$ at a time, which has the advantage that it requires a single traversal of the elements of the sparse tensor $\underline{\mathbf{X}}$, and indeed results in more efficient computation, compared to the previous solutions to MTTKRP.

In a different spirit, [PC09] propose a block PARAFAC decomposition, where they partition the original tensor into smaller sub-tensors, they distribute the computation of the sub-tensors potentially into different machines in a shared-memory architecture, and finally merge the factor matrices resulting from each individual sub-tensor by introducing multiplicative updates. The idea behind [PC09] tackles the Intermediate Data Explosion by reducing the dimensions of the tensor in a way that the intermediate data created by the ALS algorithm are no longer an issue to scalability. A potential issue, however, with [PC09] is, especially in the case of very sparse tensors, is that it is very likely that many sub-tensors will be almost entirely zero or rank-deficient, which may result in degenerate solutions.

A few years later [PFS12] introduce a parallel PARAFAC decomposition which uses biased sampling to extract sub-tensors from the data. Suppose that we have a sample of rows which is a set of indices$\mathcal{I}$, and accordingly $\mathcal{J}$ and $\mathcal{K}$ for the columns and third-mode fibers. Then, each sub-tensor $\underline{\mathbf{X}}_p$ is defined as

$$\underline{\mathbf{X}}_p = \underline{\mathbf{X}}(\mathcal{I}, \mathcal{J}, \mathcal{K})$$

The algorithm extracts many different sub-tensors $\underline{\mathbf{X}}_p$ which are in turn decomposed in parallel. In the end [PFS12] merges the individual partial results from the sub-tensors by filling-in the values in the indices of the factors that have been sampled in the first step. This has the fortuitous by-product that the resulting factors will be sparse by construction, which is very desirable both for storage and interpretability. Furthermore, [PFS12] provably guarantees that the different samples will be merged to the correct corresponding components. The fundamental difference from [PC09] is that sampling selects sub-tensors that are more likely to lead to high quality solutions. Furthermore, [PFS12] processes fewer sub-tensors than [PC09], especially when the tensor is sparse. The idea of [PFS12] is later on extended in the case of Coupled Matrix-Tensor Factorization in [PMS+14], showing significant speedups.

In the realm of Boolean tensor decompositions, [EM13] propose an algorithm which bears a few similarities with [PFS12] in the sense that it uses randomization to select dense blocks of within a tensor, and decomposes those blocks. In contrast to [PFS12], [EM13] uses random walks to identify dense blocks in a binary tensors. The authors define a graph where the nodes are the non-zero elements of the tensor, and the edges connect elements that share at least two indices (in a three-mode tensor). The blocks that the random walk finds correspond to rank-one components of a Boolean PARAFAC decomposition, thus the method returns the Boolean decomposition of the most important blocks in the data.

As we saw, [PFS12] uses sampling to reduce the dimensionality of the data, and parallelizes the decomposition. In a follow-up work [SPF14] propose an alternative scheme, where instead of sampling, they use random projection matrices $\mathbf{U}_1, \mathbf{U}_2, \mathbf{U}_3$ (not to be confused with the TUCKER matrices of [BA98]) and compress the original tensor $\underline{\mathbf{X}}$ into a smaller tensor $\underline{\mathbf{X}}_p$ as

$$\underline{\mathbf{X}}_p = \underline{\mathbf{X}} \times_1 \mathbf{U}_1 \times_2 \mathbf{U}_2 \times_3 \mathbf{U}_3.$$

As in [PFS12], they create multiple compressed tensors $\underline{\mathbf{X}}_p$ which are decomposed in parallel, and at the end, solving a least squares problem they are able, under mild conditions, to identify the true factors of $\underline{\mathbf{X}}$ up to column permutations and scaling, which is a very important theoretical guarantee of correctness. [SPF14] build upon the result of [SK12] where the authors show that one can reconstruct the original $\mathbf{A}, \mathbf{B}, \mathbf{C}$ from a single compressed replica, provided that the latent factors are sparse (requirement which is not posed by [SPF14]). Subsequent work by [RSSK14] shows how the compression of $\underline{\mathbf{X}}$ into $\underline{\mathbf{X}}_p$ can be done in a memory-efficient fashion.

In [DAK+14] we find a parallel approach which builds upon the idea of breaking down the tensor into a grid, such as in [PC09], and parallelizing the computation. The novelty in [DAK+14] is the fact that the communication between different machines that are working on separate tensor blocks is very important, especially so that machines that work on blocks that correspond to the same part of the latent factors can collaborate. Choosing the connectivity right, by using multi-layer graphs to define connectivity between machines, can speed up the ALS algorithm, while ending up in solutions that under assumptions are identifiable.

Most of the methods above, with the exception of [PFS12] and[CFC15], are focused on the "vanilla" PARAFAC decomposition, without imposing any types of constraints. In [LS15], the authors propose a paralellizable algorithm for PARAFAC based on the Alternating Direction of Multipliers Method (ADMM) [BPC+11], which can accommodate a wide variety of constraints, and is particularly well suited for distributing parts of the optimization variables over different machines.

So far, all the scalable algorithms that we have seen, either explicitly or implicitly, use the ALS algorithm. The work of [BKP+14], however, is introducing a Map/Reduce algorithm based on Distributed Stochastic Gradient Descent. The algorithm first splits the tensor into disjoint blocks, or "strata", which correspond to disjoint parameters in the factor matrices. Each stratum is distributed to a different computer, and for each stratum, the algorithm uses Stochastic Gradient Descent (SGD) updates, to estimate the parameters of the factors corresponding to that stratum. In order to correctly cover all the data, the algorithm creates different sets of disjoint blocks and iterates over those block configurations. SGD is a very efficient, stochastic algorithm, which uses a randomly selected data point at every different update, instead of using the entire data as other gradient based approaches do. As a framework SGD, and as a result [BKP+14], is very flexible and can accommodate different objective functions (e.g., KL-Divergence instead of Frobenius norm), or regularizations such as ℓ_1 norm penalties. One subtle issue to note here is that SGD is sampling from the "observed" values at random, there is an inherent assumption that 0 values in the data are considered "unobserved" or "missing", a fact that we also touch upon in Section 3.2.1.3.

Subsequently, [SK14] propose two distributed methods for Map/Reduce, one based on ALS and one based on Coordinate Descent. The ALS-based method works on sets of columns of the factor matrices, and updates entire rows at a time. Coordinate Descent on the other hand, updates individual coefficients of the factor matrices, in a column-wise fashion. The authors demonstrate that both methods scale well, both in terms of tensor dimensionality and number of non-zeros, and show that their ALS-based method is more efficient in terms of convergence speed, whereas the Coordinate Descent method offers significant memory gains.

Finally, the work of [KC12] is quite different in flavor, and draws from the domain of Relational Databases. The basic idea behind the proposed method is as follows: An N-mode tensor can be seen as a "relation" or table in a relational database, where each mode is a different column. A widely used operation in databases is the so-called "normalization", where one splits up a relation with many columns (or alternatively a high order tensor) into smaller relations, where one of those columns is shared. This is exactly what [KC12] does to split up a large and high order tensor into smaller ones, computes the PARAFAC decomposition on the "normalized" relations/tensors, and in the end, merges the factors using a "joining", another widely used database operation, which takes two relations with a column in common, and joins the remaining columns based on matching entries for that common column.

3.4.2 TUCKER

As we mention in the previous subsection, [KS08] was the first work to mention the issue of Intermediate Data Blowup. In the particular case of the TUCKER decomposition, the problem refers to the operation of line 4 of Algorithm 3.3:

$$\underline{\mathbf{Y}} \leftarrow \underline{\mathbf{X}} \times_N \mathbf{U}_N^T \cdots \times_{n+1} \mathbf{U}_{n+1}^T \times_{n-1} \mathbf{U}_{n+1}^T \cdots \times_1 \mathbf{U}_1^T$$

Algorithm	Compression	Exploits Sparsity	Sampling	Dist. / Parallel
[BA98]	✓			
[KBK05]		✓		
[BK07]		✓		
[PC09]				✓
[KC12]				✓
[KPHF12]		✓		✓
[PFS12] & Chapter 4		✓	✓	✓
[EM13]		✓	✓	✓
[DAK$^+$14]				✓
[CV14]		✓		✓
[APG$^+$14]		✓		
[BKP$^+$14]		✓	✓	✓
[SPF14] & Chapter 6	✓			✓
[LS15]				✓
[SK14]		✓		✓
[RSSK14]		✓		
[SRSK15]		✓		✓
[CFC15]	✓			
[JPKF15]		✓		✓

Table 3.1: Classification of scalable algorithms for PARAFAC with respect to the strategies they employ.

The above operation creates a series of intermediate results, first of which being $\underline{\mathbf{X}} \times_N \mathbf{U}_N^T$. If tensor $\underline{\mathbf{X}}$ has large dimensions, then this intermediate piece of data will be dense (since the TUCKER factors are dense matrices) and large, which creates the Intermediate Data Blowup problem. In order to tackle this problem, the authors propose to handle these N-mode products of the above operation element-wise for one or more modes of the tensor.

Especially in the cases where the tensor is sparse, executing the product element-wise for a given mode can be done more efficiently, since the computational complexity depends on the number of non-zero entries in the data. As a result of the above, the algorithm in [KS08] requires much less storage and is able to compute the TUCKER decomposition of very large tensors. A side-effect of handling one or more modes element-wise is that in some cases, this algorithm tends to be slower than Algorithm 3.3, however the gains in terms of memory requirements are up to 1000-fold.

Subsequently, the work of [Tso10] uses sparsification of a tensor in order to achieve scalability. In particular, the author proves that by randomly sampling non-zero entries of the tensor, and scaling appropriately, the expected approximation error does not suffer a lot. Thus, by sparsifying the tensor, the author demonstrates that we can compute the TUCKER decomposition (using either Algorithm 3.2 or Algorithm 3.3) much faster, while still capturing the variation of the full data.

In [CC10] the authors propose an extension to the matrix CUR decomposition [DKM+06] for tensors, different and more efficient from the Tensor-CUR decomposition [MMD08]; in a nutshell, CUR decomposes a matrix $\mathbf{X} \approx \mathbf{CUR}$, where $\mathbf{C}$ contains sampled columns of the $\mathbf{X}$, $\mathbf{R}$ contains sampled rows, and $\mathbf{U}$ is computed such as the squared error is minimized. In this work, the authors introduce an adaptive algorithm for selecting the rows, columns, and third-mode fibers of the tensor which can be done efficiently, resulting in the fast computation of a TUCKER-like model.

Fairly recently, [JPKF15] follow-up the work in [KPHF12], which is PARAFAC ALS for Map/Reduce, by introducing a unified decomposition framework with operations that are applicable both for PARAFAC and TUCKER decompositions. In particular, [JPKF15] introduces a scalable and distributed implementation of the N-mode product, which is essential to both PARAFAC (as [BK07] demonstrates) and TUCKER computations, by decoupling its steps in a way that is suited for the Map/Reduce framework. As an extension of [JPKF15], the same research group provides a scalable implementation of Coupled Matrix-Tensor Factorization in [JJK16].

Finally, the most recent work that is speeding up the TUCKER decomposition is by [ABK16], where the authors propose the first distributed memory implementation of Algorithms 3.2 and 3.3. They identify the basic bottlenecks algorithms, essentially the N-mode product and the computation of the R leading singular vectors of a matricized tensor (which they do by computing the eigenvectors of a symmetric matrix created from the matricized tensor). The proposed implementation is very scalable and was able to decompose (and ultimately compress, by using TUCKER) multiple terabytes of scientific data expressed as high-order tensors.

Algorithm	Exploits Sparsity	Sampling	Dist. / Parallel
[KS08]	✓		
[CC10]		✓	
[Tso10]	✓	✓	
[JPKF15]	✓		✓
[ABK16]			✓

Table 3.2: Classification of scalable algorithms for TUCKER with respect to the strategies they employ. We omit compression, since this has been an overarching theme for scaling up PARAFAC (often using TUCKER to obtain such compression).

3.4.3 H-Tucker

To the best of our knowledge, [PCVS15] is the first (and currently the only) scalable algorithm for the Hierarchical Tucker Decomposition. The main computational bottleneck, especially for very high order tensors, is the fact that the matricizations that Algorithm 3.5 is doing will have a prohibitively large dimension, since it will be the product of the sizes of all but one of the dimensions. This, in turn, makes the computation of the leading singular vectors per matricization an extremely hard problem, which makes Algorithm 3.5 unable to scale for very large and high-order tensors. The main idea behind [PCVS15] is to use a sampling approach inspired by CUR , where instead of using the very high dimensional matricizations, the authors sample a small number of columns (and make sure that the sampling is consistent across matricizations). Assuming that the original tensor is very sparse, the algorithm in [PCVS15] ensures that all the data in the algorithm, both intermediate and results, are sparse, thus making the computations much faster and scalable.

3.5 Conclusions

Tensors decompositions are very versatile and powerful tools, ubiquitous in data mining applications. As we saw, they have been successfully integrating in a rich variety of real-world applications, and due to the fact that they can express and exploit higher order relations in the data, they tend to outperform approaches that ignore such structure. Furthermore, recent advances in scaling up tensor decompositions has employed practitioners with a strong arsenal of tools that can be applied to many big multi-aspect data problems.

Part I

Algorithms - Scalability and Efficiency

Chapter based on material published in [PFS12] and [PFS15].

Chapter 4

ParCube: Sparse Parallelizable PARAFAC Decomposition

Exploiting sparsity and sampling for paralellizing and speeding up PARAFAC.

How can we efficiently decompose a tensor into sparse factors, when the data does not fit in memory? In this Chapter, we propose ParCube, a new, triple-sparse and highly parallel method for speeding up tensor decompositions that is well-suited to producing sparse approximations. Experiments with even moderately large data indicate over 90% sparser outputs and 14 times faster execution, with approximation error close to the current state of the art irrespective of computation and memory requirements.

4.1 Introduction

Tensors and tensor decompositions have recently attracted considerable attention in the data mining community. With the constantly increasing volume of today's multi-dimensional datasets, tensors are often the 'native' format in which data is stored, and tensor decompositions the natural modeling toolset - albeit still suffering from major scalability issues. The state of the art toolboxes for tensors [BK+15, AB00] still operate on main memory and cannot possibly handle disk-resident tensor datasets, in the orders of millions or billions of non-zeros.

Motivated by the success of random sampling - based matrix algorithms such as [DKM+06], it is natural to ask whether we can we use similar tools in the case of tensors. Is it possible 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 costs? There exists important work on how to use sampling in order to achieve a sparse matrix decomposition, the CUR decomposition [DKM+06]; this method has also been extended in order to handle tensors [MMD06]. However, both these methods are tied to a specific decomposition, while we desire to disconnect sampling from the specific decomposition that follows.

This chapter introduces ParCube, a fast and parallelizable method for speeding up tensor decompositions by leveraging random sampling techniques. A nice side-benefit of our algorithm is its natural tendency to produce sparse outer-product approximations, i.e., the model-synthesized approximation of the given tensor data is naturally very sparse, 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. Figure 4.1 shows a high-level overview of how ParCube works (described in detail in Section 4.2.

Our core contribution is in terms of the merging algorithm that collects the different 'punctured' decompositions and combines them into one overall decomposition in an efficient way. We provide theoretical guarantees for the correctness of our approach.

Furthermore, we apply ParCube on four different, real datasets, reporting our discoveries and demonstrating the remarkable flexibility and versatility of Tensor Analysis as a Data Mining tool.

An earlier version of the present work has appeared in the proceedings of ECML-PKDD 2012 [PFS12]. In this extended version, in addition to the contributions of [PFS12], we provide a thorough experimental analysis of the algorithm, investigating scalability of ParCube in a variety of scenarios; additionally, we complement our description of ParCube with an intuitive explanation behind the main idea, and

In order to promote and encourage reproducibility of the results, we provide a very efficient parallel implementation which we make publicly available at `http://www.cs.cmu.edu/~epapalex/src/parCube.zip`.

4.2 Proposed Method

In this section we introduce ParCube, a novel, parallel algorithm for Parafac decomposition.

A scalar is denoted by a lowercase, italic letter, e.g. x. A column vector is denoted by a lowercase, boldface letter, e.g. $\mathbf{x}$. A matrix is denoted by an uppercase, boldface letter, e.g. $\mathbf{X}$. A three-way tensor is denoted by $\underline{\mathbf{X}}$. Let $\mathcal{I}$ be a set of indices, e.g. $\mathcal{I} = \{1, 4, 7\}$; then, $\mathbf{a}(\mathcal{I})$ denotes $\{\mathbf{a}(1), \mathbf{a}(4), \mathbf{a}(7)\}$; $\mathbf{a}(:)$ spans all the elements of $\mathbf{a}$. This notation naturally extends to matrices and tensors, i.e., $\mathbf{A}(\mathcal{I}, :)$ comprises all columns of $\mathbf{A}$ restricted to rows in $\mathcal{I}$. By $NNZ(\;)$ we denote the number of non-zeros

Before we proceed with the description of ParCube, first we define a class of algorithms that we desire our method to fall under:

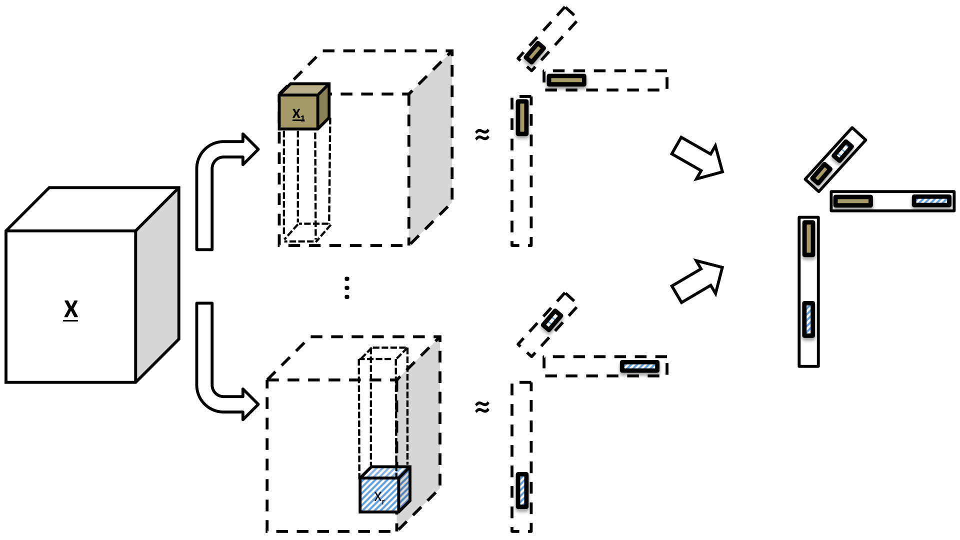


Figure 4.1: The main idea behind PARCUBE: Using biased sampling, extract small representative sub-sampled tensors, decompose them in parallel, and carefully merge the final results into a set of sparse latent factors.

Definition 6. Triple-sparse: An algorithm is triple-sparse when 1) the input of the algorithm is sparse, 2) the intermediate data during the lifetime of the algorithm is sparse, and 3) the final output of the algorithm is sparse.

In the above definition, the input of the algorithm need not necessarily be sparse; however, a triple-sparse algorithm still satisfies the second and third requirement, by operating on a sparse, representative subset of the data. We, thus, call PARCUBE, a *triple-sparse* algorithm. We have to note that we consider the intermediate data that the algorithm manipulates to be sparse *with respect to the original data*, and not necessarily sparse with respect to the sample size. In addition to storage and efficiency, a major benefit of sparsity is *interpretability*: when most of the coefficients of the factors are zero, it is easier for a person to inspect the non-zeros and identify patterns in the results. Additionally, having sparse latent factors has been shown to be equivalent to higher order co-clustering [PS11, PSB13] which is a very useful and highly interpretable tool in exploratory data mining.

In addition to PARCUBE being a triple-sparse algorithm, we design the method with the following three main goals in mind: **G1**: Relative simplicity, speed, and parallelizable execution; **G2**: Ability to yield sparse latent factors and a sparse tensor approximation; and **G3**: provable correctness in merging partial results, under appropriate conditions.

4.2.1 Sampling for PARCUBE

The first step of PARCUBE is to sample a very high dimensional tensor and use the sampled tensor in lieu of the original one, bearing three important requirements in mind: **R1** The need to significantly reduce dimensionality; **R2** The desire that sampling should be decomposition-independent - we should be able to apply any decomposition we desire *after* sampling, and be able to extrapolate from that; and **R3**: Sampling should maintain linear complexity on the number of non-zero entries.

The first thing that comes to mind in order to satisfy requirement **R1** is to take a uniform random sample of the indices of each mode, i.e., take a uniform random sample of the index sets $\{1 \cdots I\}$, $\{1 \cdots J\}$, and $\{1 \cdots K\}$. However, this naive approach may not adequately preserve the data distribution, since the random index samples may correspond to entirely arbitrary rows/columns/fibers of the tensor. We performed initial tests using this naive method, and the results were consistently worse than the proposed method's. We thus propose to do *biased* sampling: If we, somehow, determine a measure of importance for each index of each mode, then we may sample the indices using this measure as a sampling weight/probability. For the purposes of this work, let us assume that our tensor $\underline{\mathbf{X}}$ has *non-negative* entries (which is the case in huge variety of data mining applications); if we were to deal with tensors containing real values, we should consider the element-wise absolute value of the tensor for the notions that we introduce in the sequel.

A reasonable measure of importance is the marginal sum of the tensor for each mode, however we can also use the sum of squares of instead, as a measure of *energy*. Namely, the measure of importance for the indices of the first mode is defined as: $\mathbf{x}_a(i) = \sum_{j=1}^{J} \sum_{k=1}^{K} \underline{\mathbf{X}}(i,j,k)$ for $i = 1 \cdots I$.

Similarly, we define the following importance measures for modes 2 and 3:

$$\mathbf{x}_b(j) = \sum_{i=1}^{I} \sum_{k=1}^{K} \underline{\mathbf{X}}(i,j,k), \mathbf{x}_c(k) = \sum_{i=1}^{I} \sum_{j=1}^{J} \underline{\mathbf{X}}(i,j,k)$$

for $j = 1 \cdots J$ and $k = 1 \cdots K$.

Intuitively, if $\mathbf{x}_a(i)$ is high for some i, then we would desire to select this specific index i for our sample with higher probability than others (which may have lower $\mathbf{x}_a$ value). This is the very idea behind PARCUBE: We sample the indices of each mode of $\underline{\mathbf{X}}$ without replacement, using $\mathbf{x}_a$, $\mathbf{x}_b$ and $\mathbf{x}_c$ to bias the sampling probabilities.

We define s to be the sampling factor, i.e. if $\underline{\mathbf{X}}$ is of size $I \times J \times K$, then $\underline{\mathbf{X}}_s$ derived by PARCUBE will be of size $\frac{I}{s} \times \frac{J}{s} \times \frac{K}{s}$. We may also use different sampling factors for each mode of the tensor, without loss of generality.

In order to obtain the sample we 1) Compute set of indices $\mathcal{I}$ as random sample without replacement of $\{1 \cdots I\}$ of size I/s with probability $p_{\mathcal{I}}(i) = \mathbf{x}_a(i) / \sum_{i=1}^{I} \mathbf{x}_a(i)$. 2) Compute set of indices $\mathcal{J}$ as random sample without replacement of $\{1 \cdots J\}$ of size J/s with probability $p_{\mathcal{J}}(j) = \mathbf{x}_b(j) / \sum_{j=1}^{J} \mathbf{x}_b(j)$. 3) Compute set of indices $\mathcal{K}$ as random sample without replacement of $\{1 \cdots K\}$ of size K/s with probability $p_{\mathcal{K}}(k) = \mathbf{x}_c(k) / \sum_{k=1}^{K} \mathbf{x}_c(k)$.

The ParCube method defines a means of sampling the tensor across all three modes, without relying on a specific decomposition or a model. Therefore, it satisfies requirement **R3**. Algorithm 4.1 provides an outline of the sampling for ParCube.

Lemma 4.1.:
The computational complexity of Algorithm 4.1 is linear in the number of non zero elements of $\underline{\mathbf{X}}$.

Proof. Suppose we have a representation of $\underline{\mathbf{X}}$ in quadruplets of the form (i, j, k, v) where $\underline{\mathbf{X}}(i, j, k) = v$, for $v \neq 0$ and $v \in NNZ(\underline{\mathbf{X}})$. For each of these quadruplets, we may compute the density vectors as:

$$\mathbf{x}_a(i) = \mathbf{x}_a(i) + v, \mathbf{x}_b(j) = \mathbf{x}_b(j) + v, \mathbf{x}_c(k) = \mathbf{x}_c(k) + v$$

This procedure requires 3 $O(1)$ additions per element v, therefore the total running time is $O(NNZ(\underline{\mathbf{X}}))$. ■

By making use of the above Lemma, and noticing that sampling of the elements, after having computed the densities of each mode is a linear operation on the number of non-zeros, we conclude that requirement **R3** is met, i.e. our computation of the biases and biased sampling are linear on the number of non-zeros. Furthermore, sampling pertains to Goal **G1** which calls for a fast algorithm.

Algorithm 4.1: BiasedSample

```
Input: Original tensor $\underline{\mathbf{X}}$ of size $I \times J \times K$, sampling factor $s$.
Output: Sampled tensor $\underline{\mathbf{X}}_s$, index sets $\mathcal{I}, \mathcal{J}, \mathcal{N}$.
1: Compute
   $\mathbf{x}_a(i) = \sum_{j=1}^{J} \sum_{k=1}^{K} \underline{\mathbf{X}}(i,j,k)$, $\mathbf{x}_b(j) = \sum_{i=1}^{I} \sum_{k=1}^{K} \underline{\mathbf{X}}(i,j,k)$, $\mathbf{x}_c(k) = \sum_{i=1}^{I} \sum_{j=1}^{J} \underline{\mathbf{X}}(i,j,k)$.
2: Compute set of indices $\mathcal{I}$ as random sample without replacement of $\{1 \cdots I\}$ of size
   $I/s$ with probability $p_{\mathcal{I}}(i) = \mathbf{x}_a(i) / \sum_{i=1}^{I} \mathbf{x}_a(i)$. Likewise for $\mathcal{J}, \mathcal{K}$.
3: Return $\underline{\mathbf{X}}_s = \underline{\mathbf{X}}(\mathcal{I}, \mathcal{J}, \mathcal{K})$.
```

4.2.2 Non-negative Parafac decomposition using ParCube

Now, let us demonstrate how to apply ParCube in order to scale up the popular Parafac decomposition, with non-negativity constraints. We choose to operate under the non-negativity regime since the vast majority of applications of interest naturally impose this type of constraint.

Algorithm 4.2 demonstrates the most basic approach in which one extracts a sample from the original tensor, runs the PARAFAC decomposition on that (significantly) smaller tensor and then redistributes the factor vectors to their original positions, according to the sampled indices $\mathcal{I}, \mathcal{J}, \mathcal{K}$. Note that many of the coefficients of the resulting PARAFAC factor matrices will be exactly zero, since their corresponding indices will not be included in the sample and consequently, they will not receive an updated value. This implies that a natural by-product of our approach is *sparsity on the factors by construction*, thereby satisfying Goal **G2**.

However, Algorithm 4.2 relies on a sole sample of the tensor and it might be the case that some significant portions of the data, depending on the sampling factor and the data distribution, may be left out. To that end, we introduce Algorithm 4.3 which is our main contribution. Algorithm 4.3 generates many samples and correctly combines them, in order to achieve better extraction of the true latent factors of the data tensor.

The key idea behind Algorithm 4.3 is the method by which all the different samples are combined in order to output the decomposition matrices; more specifically, intuitively we enforce all the different samples to have a common set of indices $\mathcal{I}_p, \mathcal{J}_p, \mathcal{K}_p$, where $p \in [0, 1]$ is a fraction of the sampled indices per mode. For example, for a mode of size I and sampling factor s, the common set of indices will be of size $\mathcal{I}_p$ is pI/s. This p fraction of indices is selected to be the indices with the highest density, as indicated by the weights that we compute. After we select these common indices, the rest of the sampling is being conducted on the *remaining* indices. Having this common basis, we are able to combine the samples using Algorithm 4.4. In section 4.2.3, we elaborate on the proposed merging scheme, and provide an illustrative example.

Note that the generation of the r distinct samples of $\underline{\mathbf{X}}$, as well as the PARAFAC decomposition of each of them may be carried out in parallel; thus satisfying Goal **G1**. Regarding Goal **G3**, note that correctness of the merge operation requires certain conditions; it cannot be guaranteed when the individual random samples do not satisfy PARAFAC identifiability conditions, or when the common piece that is used as a reference for merging is too small (p is too low). Proposition 4.1 provides a first correctness result for our merging algorithm.

4.2.3 Merging explained

The component permutation problem

Suppose we want to merge the partial factor matrices $\mathbf{A}_i$ $i = 1 \cdots r$ into the full-sized factor matrix $\mathbf{A}$. The ordering of the PARAFAC components is arbitrary, since the PARAFAC decomposition is unique up to scaling and component permutations. Since any ordering is good, we have to, arbitrarily, agree on an ordering for the components/columns of $\mathbf{A}$. A problem that arises when we are about to merge the partial factors $\mathbf{A}_i$ into $\mathbf{A}$ is the fact that each $\mathbf{A}_i$ has its own arbitrary ordering of columns which, sometimes, is not consistent for all i. We, thus, have to first agree on a single ordering, and then permute the columns of all $\mathbf{A}_i$ such that they obey that ordering.

The key idea

Key to identifying the correct correspondence of columns between different $\mathbf{A}_i$ is the common set of sampled indices $\mathcal{I}_p, \mathcal{J}_p, \mathcal{K}_p$. By fixing these indices, we force the rows of matrices $\mathbf{A}_i$ that correspond to indices $\mathcal{I}_p$ to be approximately the same, and accordingly for the rows of $\mathbf{B}_i$ and $\mathcal{J}_p$, as well as $\mathbf{C}_i$ and $\mathcal{K}_p$. We will elaborate more in subsection 4.2.4 about the conditions that need to be met, in order for the rows that correspond to the same subset of indices to be approximately equal, but we may assume that this is the case, for the purposes of explaining the merging algorithm (Algorithm 4.4). It is important to note here that we normalize every column of $\mathbf{A}_i$ in such a way that the $\|\mathbf{A}(\mathcal{I}_p, j)\|_F = 1$ for $j = 1 \cdots F$ (we do the same for $\mathbf{B}_i$ and $\mathbf{C}_i$).

The basic idea of Algorithm 4.4 is the following: We arbitrarily choose the columns of $\mathbf{A}_1$ as the reference ordering. After doing that, we update the columns of $\mathbf{A}$ (the final matrix), which originally contains all zero values, using the values of $\mathbf{A}_i$. The way we update is described in Algorithm 4.2. In the next iteration of Algorithm 4.4, we first need to permute the columns of $\mathbf{A}_2$ so that they match the (arbitrary) ordering of the columns of $\mathbf{A}_1$ that we decided upon. In order to do that, we take the inner products of combinations of the common parts of the columns of $\mathbf{A}_1$ and $\mathbf{A}_2$. Because of the way we have normalized, the parts of the matrices that correspond to the common set of indices will have unit norm; thus, for the matching pair, the inner product will be approximately equal to 1, whereas for the rest of the pairs it will be close to 0. We prove this claim in subsection 4.2.4. After we establish the correct ordering, we update only the non-zero coefficients of $\mathbf{A}$ using $\mathbf{A}_2$. We choose to update only the non-zero values, since averaging values that happen to correspond to different samples was not retaining the correct scaling of the factors.

After we establish the correct correspondence for the columns of $\mathbf{A}_i$, we can apply the same permutation to the columns of $\mathbf{B}_i$ and $\mathbf{C}_i$, instead of computing the correspondence seperately. In addition to the factor matrices, we have to also reorder the $\boldsymbol{\lambda}_i$ vector at every merging step.

Illustrative example & discussion

An illustrative example of our merging scheme, for two partial factor matrices, is shown in Fig. 4.2. Here, we describe the merging procedure. Say that the small matrices shown on the leftmost part the figure are the $\mathbf{A}_i$ $i = 1 \cdots r$ matrices (where $r = 2$ in the example). Each color corresponds to a distinct latent component; different shades of the same color denote the fact that two vectors belong to the same rank-one component of the non-sampled tensor, however they correspond to a different set of sampled indices. Notice that there are component permutations across matrices which need to be resolved in order to merge correctly. Without loss of generality, assume that the upper part of each component is the common part, defined by the shared sample of indices. The common part is denoted by a dark shade of the color of each component, in Fig. 4.2. Then, Algorithm 4.4 will do the following steps: starting from $\mathbf{A}_1$, it will redistribute the values of the factors to the original index space. The ordering of components imposed

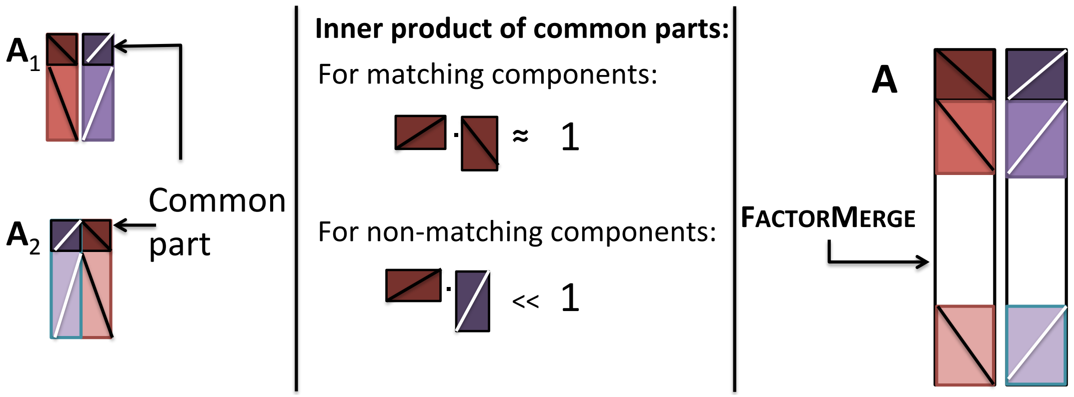


Figure 4.2: Example of merging two partial factor matrices to the final matrix, while accounting for potential column permutations. (Best viewed in color)

by $\mathbf{A}_1$ (shown in different colors in Fig. 4.2) is the order that Algorithm 4.4 will impose to the rest of the partial results. In Fig. 4.2, the second partial factor matrix $\mathbf{A}_2$ has a different ordering of the last two components, therefore the algorithm will use the common part in order to identify this discrepancy, and reorder the columns of $\mathbf{A}_2$ before merging its values to the final result. The algorithm proceeds this way, until all partial matrix columns have been reordered to match the ordering of $\mathbf{A}_1$.

A fairly subtle issue that arises is how to overcome scaling disparities between factors coming from two different samples. Key here, as described in line 5 of Algorithm 4.3, is to counter-scale the two merge candidates, using only the norms of the common parts indexed by $\mathcal{I}_p, \mathcal{J}_p, \mathcal{K}_p$; by doing so, the common parts will be scaled to unit norm, and the rest of the vectors will also refer to the correct, same scaling, thereby effectively resolving scaling correspondence.

Finally, we must note that our merging scheme is equivalent to the very well know Hungarian Algorithm [Kuh55], used to efficiently solve combinatorial assignment problems.

Algorithm 4.2: Basic PARCUBE for Non-negative PARAFAC

```
Input: Tensor $\underline{\mathbf{X}}$ of size $I \times J \times K$, number of components $F$, sampling factor $s$.
Output: Factor matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$ of size $I \times F$, $J \times F$, $K \times F$ respectively.
1: Run BIASEDSAMPLE ($\underline{\mathbf{X}}$, $s$) (Algorithm 4.1) and obtain $\underline{\mathbf{X}}_s$ and $\mathcal{I}, \mathcal{J}, \mathcal{K}$.
2: Run Non-Negative PARAFAC ($\underline{\mathbf{X}}_s$, $F$) and obtain $\mathbf{A}_s, \mathbf{B}_s, \mathbf{C}_s$ of size $I/s \times F$, $J/s \times F$ and $K/s \times F$.
3: $\mathbf{A}(\mathcal{I}, :) = \mathbf{A}_s$, $\mathbf{B}(\mathcal{J}, :) = \mathbf{B}_s$, $\mathbf{C}(\mathcal{K}, :) = \mathbf{C}_s$
```

4.2.4 Correctness

In the following lines, we prove that when we have multiple repetitions, the FACTORMERGE Algorithm is going to find the right correspondence between the compo-

Algorithm 4.3: PARCUBE for Non-negative PARAFAC with repetition

```
Input: Tensor $\underline{\mathbf{X}}$ of size $I \times J \times K$, number of components $F$, sampling factor $s$, number
    of repetitions $r$.
Output: PARAFAC factor matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$ of size $I \times F$, $J \times F$, $K \times F$ respectively and
    vector $\boldsymbol{\lambda}$ of size $F \times 1$ which contains the scale of each component.
1: Initialize $\mathbf{A}, \mathbf{B}, \mathbf{C}$ to all-zeros.
2: Randomly, *using mode densities as bias*, select a set of $100p\%$ ($p \in [0,1]$) indices
   $\mathcal{I}_p, \mathcal{J}_p, \mathcal{K}_p$ to be common across all repetitions.
3: for $i = 1 \cdots r$ do
4:     Run Algorithm 4.2 with sampling factor $s$, using $\mathcal{I}_p, \mathcal{J}_p, \mathcal{K}_p$ as a common reference
       among all $r$ different samples and obtain $\mathbf{A}_i, \mathbf{B}_i, \mathbf{C}_i$. The sampling is made on the
       set difference of the set of all indices and the set of common indices.
5:     Calculate the $\ell_2$ norm of the columns of the common part: $\mathbf{n}_a(f) = \|\mathbf{A}_i(\mathcal{I}_p, f)\|_2$,
       $\mathbf{n}_b(f) = \|\mathbf{B}_i(\mathcal{J}_p, f)\|_2$, $\mathbf{n}_c(f) = \|\mathbf{C}_i(\mathcal{K}_p, f)\|_2$ for $f = 1 \cdots F$. Normalize columns of
       $\mathbf{A}_i, \mathbf{B}_i, \mathbf{C}_i$ using $\mathbf{n}_a, \mathbf{n}_b, \mathbf{n}_c$ and set $\boldsymbol{\lambda}_i(f) = \mathbf{n}_a(f)\mathbf{n}_b(f)\mathbf{n}_c(f)$. Note that the common
       part will now be normalized to unit norm.
6: end for
7: $\mathbf{A}$ =FACTORMERGE ($\mathbf{A}_i$)
8: $\mathbf{B}$ =FACTORMERGE ($\mathbf{B}_i$),$\mathbf{C}$ =FACTORMERGE ($\mathbf{C}_i$) without computing the ordering
   from scratch. Use the ordering obtained for the $\mathbf{A}_i$.
9: Apply the same ordering to $\boldsymbol{\lambda}_i$.
10: $\boldsymbol{\lambda}$ = average of $\boldsymbol{\lambda}_i$.
```

Algorithm 4.4: FACTORMERGE

```
Input: Factor matrices $\mathbf{A}_i$ of size $I \times F$ each, where $i = 1 \cdots r$, and $r$ is the number of
    repetitions, $\mathcal{I}_p$: set of common indices.
Output: Factor matrix $\mathbf{A}$ of size $I \times F$.
1: Set $\mathbf{A} = \mathbf{A}_1$
2: for $i = 2 \cdots r$ do
3:     for $f_1 = 1 \cdots F$ do
4:         for $f_2 = 1 \cdots F$ do
5:             Compute similarity $\mathbf{v}(f_2) = (\mathbf{A}(\mathcal{I}_p, f_2))^T (\mathbf{A}_i(\mathcal{I}_p, f_1)))$
6:         end for
7:         $c = \arg\max_{c'} \mathbf{v}(c')$
8:         Update only the zero entries of $\mathbf{A}(:, c)$ using vector $\mathbf{A}_i(:, f_1)$.
9:     end for
10: end for
```

nents of the intermediate results, and thus, improve the approximation of the original data.

Proposition 4.1.:
Let $(\mathbf{A}, \mathbf{B}, \mathbf{C})$ be the PARAFAC decomposition of $\underline{\mathbf{X}}$, and assume that $\mathbf{A}(\mathcal{I}_p, :)$ ($\mathbf{A}$ restricted

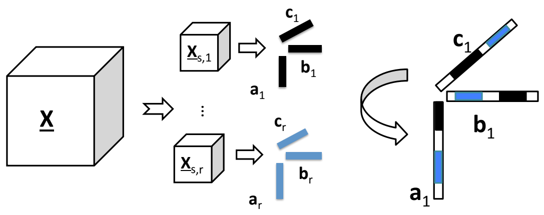


Figure 4.3: Example of rank-1 PARAFAC using PARCUBE (Algorithm 4.3). The procedure described is the following: Create r independent samples of $\underline{\mathbf{X}}$, using Algorithm 4.1. Run the PARAFAC- ALS algorithm for $K = 1$ and obtain r triplets of vectors, corresponding to the first component of $\underline{\mathbf{X}}$. As a final step, combine those r triplets, by distributing their values to the original sized triplets, as indicated in Algorithm 4.3.

to the common I-mode reference rows) is such that any two of its columns are linearly independent; and likewise for $\mathbf{B}(\mathcal{J}_p, :)$ and $\mathbf{C}(\mathcal{K}_p, :)$. Note that if $\mathbf{A}(\mathcal{I}_p, :)$ has as few as 2 rows ($|\mathcal{I}_p| \geq 2$) and is drawn from a jointly continuous distribution, this requirement on $\mathbf{A}(\mathcal{I}_p, :)$ is satisfied with probability 1. Further assume that each of the sub-sampled models is identifiable, and the true underlying rank-one (punctured) factors are recovered, up to permutation and scaling, from each sub-sampled dataset. Then Algorithm 4.4 is able to merge the factors coming from the different samples of the tensor correctly, i.e., is able to find the correct correspondence between the columns of the factor matrices $\mathbf{A}_i, \mathbf{B}_i, \mathbf{C}_i$.

Proof. Consider the common part of the $\mathbf{A}$-mode loadings recovered from the different sub-sampled versions of $\underline{\mathbf{X}}$: under the foregoing assumptions, the $\mathbf{A}_i(\mathcal{I}_p, :)$ will be permuted and column-scaled versions of $\mathbf{A}(\mathcal{I}_p, :)$. After scaling *the common part* of each column to unit norm, Algorithm 4.4 seeks to match the permutations by maximizing correlation between pairs of columns drawn from $\mathbf{A}_i(\mathcal{I}_p, :)$ and $\mathbf{A}_j(\mathcal{I}_p, :)$. From the Cauchy-Schwartz inequality, correlation between any two unit-norm columns is ≤ 1, and equality is achieved only when the correct columns are matched, because any two distinct columns of the underlying $\mathbf{A}(\mathcal{I}_p, :)$ are linearly independent. Furthermore, by normalizing the scales of the matched columns to equalize the norm of the common reference part, the insertions that follow include the correct scaling too. This shows that Algorithm 4.4 works correctly in this case. ■

The above proposition serves as a sanity check for correctness. In reality, there will be noise and other imperfections that come into play, implying that the punctured factor estimates will at best be approximate. This implies that a larger common sample size ($|\mathcal{I}_p| \geq 2$, $|\mathcal{J}_p| \geq 2$, $|\mathcal{K}_p| \geq 2$) will generally help Algorithm 4.4 to correctly merge the pieces coming from the different samples. We have carried out extensive experiments verifying that Algorithm 4.4 works well in practice, under common imperfections. A reasonable value for p is about 10-20 percent of the sampled indices, depending on the application at hand. Those experiments also suggest that increasing the number of samples, r, reduces the PARAFAC approximation error.

A good rule of thumb on selecting the number of repetitions r is to set it equal to double the sampling factor, since this will, empirically, allow for PARCUBE to explore most of the variation in the data. The exact values for s, r depend on the sparsity of the original tensor; if the tensor is highly sparse, then only a few, small samples may suffice. In Chapter 6 we propose PARACOMP, a formal extension of PARCUBE, were we are able to prove identifiability of the decomposition, as well as give precise guidelines on the size of the sample and the number of repetitions.

4.2.5 Parallel Algorithm

A great advantage of the proposed PARCUBE method is the fact that on its first phase, it produces r independent tensors which are significantly smaller in size. Each one of those r tensors, 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). In other words, in our parallel implementation of PARCUBE, lines 3-6 of Algorithm 4.3 are executed entirely in parallel. In the case that a machine has less than r cores/workers (say w), then w decompositions are carried out in parallel at any given point in time, until the number of repetitions is met.

4.2.6 On Sparsity

As we outline in the Introduction as well as in the requirements for the algorithm, PARCUBE produces factors which are sparse. In fact, at any given point in time throughout the lifetime of the algorithm, PARCUBE operates on a sub-sampled portion of the data, and hence, operates on sparse data. This is not generally true for tensor decomposition methods: for instance, the ALS algorithm for PARAFAC, which iteratively computes estimates of the factor matrices, operates on *dense* data, even if the original data and the true latent factors are sparse, since least squares estimates tend to be dense.

Typically, sparsity in the factors is obtained through regularization using the ℓ_1 norm (e.g., [PSB13]), as a convex relaxation of the ℓ_0 norm. Contrary to this line of work, PARCUBE achieves sparsity in a more direct way, by ignoring a portion of the parameters altogether: if an index is not sampled by PARCUBE, then the corresponding value of any factor for that index will be zero.

The nature of PARCUBE's sparsity is approximate, and it comes as a side benefit of the sampling that PARCUBE uses. We must note that if the number of sampled tensors is rather small, in the sense that they capture only a small part of the variation of the data, then there will be parameters in the factors that will be left zero, even though the optimal solution to the problem (minimizing the ℓ_0 norm of the factors) would possibly yield a non-zero value for them. However, if we increase the number of independent sampled tensors that we decompose (i.e. parameter r), as we empirically demonstrate in Section 4.3.2, PARCUBE's solution will converge to the solution that directly optimizes for sparsity.

4.2.7 Extension to other models

Even though the focus of the present chapter is the PARAFAC decomposition, the same methodology can be applied in order to accelerate and parallelize other tensor decomposition models. For instance, in Chapter 5, we illustrate how the same principles can help accelerate the Coupled Matrix-Tensor Factorization (CMTF).The CMTF model is very similar to the PARAFAC model, and thus, our algorithms can carry through without loosing the correctness guarantees.

On the other hand, extending PARCUBE to models such as TUCKER is not straightforward. In Section 4.2.4 we invoke the uniqueness of the PARAFAC factors in order to show that the merging will be correct; however, TUCKER is highly non-unique, and therefore we cannot apply the same claim that PARCUBE will work correctly. This is not to say, however, that the key concepts behind PARCUBE cannot be used for TUCKER, but simply that this needs to be done carefully, in light of the differences of TUCKER from PARAFAC.

4.3 Experimental Evaluation

In this section we provide experimental evaluation of our proposed method. First, we evaluate the performance of PARCUBE, compared to the current state of the art for handling sparse tensors in Matlab, i.e. the Tensor Toolbox for Matlab [BK+15]. Since our algorithm, by construction, tends to output sparse factors, we also evaluate the validity of that claim by comparing the degree of sparsity of the output to the one given by the Tensor Toolbox and the one given by PARAFAC-SLF [PSB13], which is the state of the art for PARAFAC decompositions with sparsity on the latent factors. The results of Section 4.3.1 were measured on a 2.7 GHz Intel Core i5 with 4GB of RAM.

Additionally, we evaluate how PARCUBE scales as the input and the parameter size grows, the benefits of executing PARCUBE in parallel, as well as how PARCUBE compares against TUCKER compression accelerated PARAFAC decomposition. The aforementioned experiments correspond to Sections 4.3.3 - 4.3.6 and were carried out on a machine with 4 Intel Xeon E74850 2.00GHz, and 512Gb of RAM.

Finally, in Section 4.4 we apply our approach in order to analyze real datasets presented in Table 4.1.

Name	**Description**	**Dimensions**	**NNZ**
ENRON [ENR14]	(sender, recipient, month)	$186 \times 186 \times 44$	9838
LBNL [PAB+05]	(src, dst, port #)	$65170 \times 65170 \times 65327$	27269
Facebook [VMCG09]	(wall owner, poster, day)	$63891 \times 63890 \times 1847$	737778
NELL [RTW16]	(noun-phrase, noun-phrase, context)	$14545 \times 14545 \times 28818$	76879419

Table 4.1: Datasets analyzed

We implemented PARCUBE in Matlab and Java, and we make it *publicly available*[1]. We

[1]Download PARCUBE at www.cs.cmu.edu/~epapalex/src/parCube.zip

furthermore use the Tensor Toolbox for Matlab [BK+15] as our core PARAFAC decomposition implementation. In our experiments, we use both synthetic and real data. The real dataset we use is ENRON [ENR14], a $186 \times 186 \times 44$ tensor of (sender, recipient, month) with 9838 non-zero entries, coming from the Enron e-mail exchange dataset.

4.3.1 Performance & Speedup Evaluation

In the following lines, we evaluate the performance of PARAFAC using PARCUBE (Algorithm 4.2). As a performance metric, we use the relative cost of the PARAFAC model, i.e. the cost of the model using our sampling approach, divided by the cost of fitting a PARAFAC model using the original tensor. As a reminder, we refer the reader to Equation 2.1 for the approximation cost of PARAFAC. In Fig. 4.4, we measure the relative cost as a function of the speedup incurred by using our PARCUBE, for different values of the sampling factor; this experiment was carried out on $100 \times 100 \times 100$ randomly generated, synthetic tensors, as we required full control over the true number of components and the degree of sparsity for each component. We did 50 iterations of the experiment, and we report the means. We observe that even for a relatively high sampling factor, the relative cost is very good, and can be further improved using more parallel repetitions which will not harm the speedup achieved.

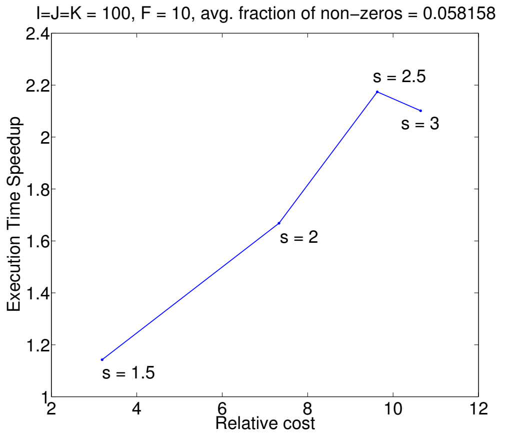
In Fig. 4.5, we show the relative cost using the ENRON dataset, for various numbers of repetitions (i.e. distinct samples). We see, in this case, that as the number of repetitions increases, the approximation improves, as expected, from our theoretical result. In both cases of Fig. 4.5, the approximation error improves as the number of repetitions r increases, as expected from our theoretical analysis of Section 4.2.4.

4.3.2 Factor Sparsity Assessment

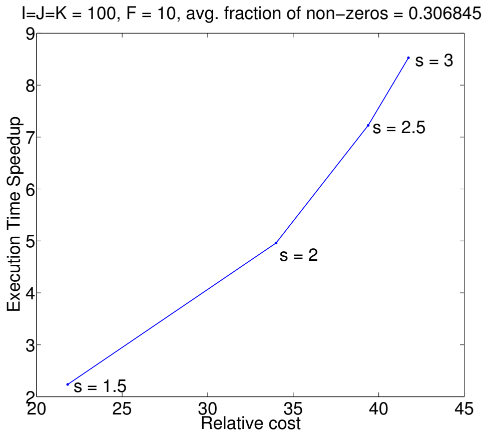
In Fig.4.6, we measure the relative output size (i.e. the relative degree of sparsity) between PARCUBE and Tensor Toolbox non-negative PARAFAC. As before, we carried out 50 iterations of the experiment and report mean values. The output size is simply defined as $NNZ(\mathbf{A}) + NNZ(\mathbf{B}) + NNZ(\mathbf{C})$, which clearly reflects the degree of sparsity in the decomposition factors. We observe that PARCUBE yields 90% sparser results than plain PARAFAC, while maintaining the same approximation error. This empirically shows that sparsity introduced by sampling in PARCUBE, albeit unconventional, produces meaningful representations of the data.

In Fig. 4.7 we measure the relative output size between PARCUBE and PARAFAC-SLF, as a function of the sampling factor s, for different values of the sparsifying parameter λ used by PARAFAC-SLF [PS11, PSB13] [2]. This further provides evidence on the validity of the sparsity introduced by PARCUBE.

[2]Code is available at http://www.cs.cmu.edu/~epapalex/src/PARAFAC_SLF.zip

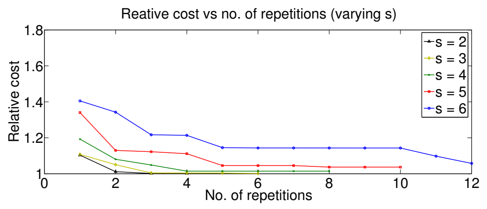


(a)

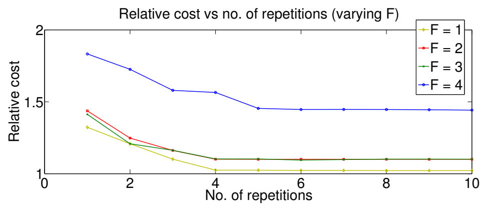


(b)

Figure 4.4: **ParCube is faster than ALS-Parafac**: Speedup vs Relative cost (ParCube/ ALS-Parafac) for 1 repetition, for varying sampling factor and different degrees of sparsity. We observe that even for a relatively high sampling factor, we get relatively good relative cost, which may be further improved using repetition. Key here is that by using repetition, because this procedure may be carried out in parallel, we may improve the accuracy and maintain similar speedup.

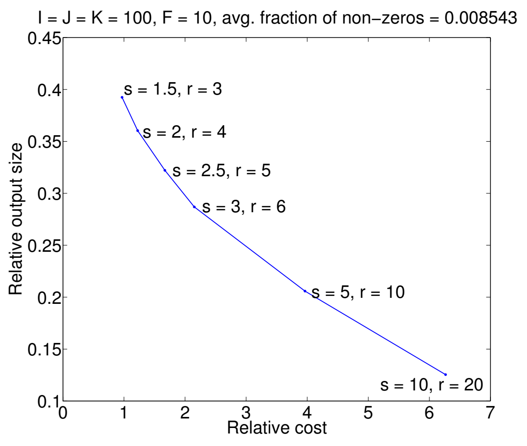


(a) ENRON: Relative error vs No. of repetitons (varying s)

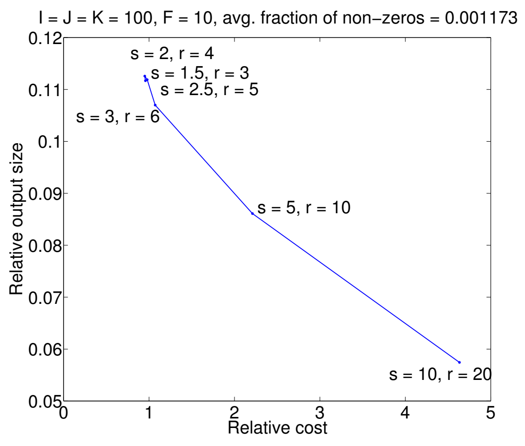


(b) ENRON: Relative error vs No. of repetitons (varying F)

Figure 4.5: **ParCube reduces the Parafac approximation error**: (a) Approximation cost vs number of repetitions for varying s, where $r = 2s$ (b) Approximation cost vs number of repetitions for varying F and fixed $s = 5$. In both cases, the approximation improves as r increases, as expected



(a)



(b)

Figure 4.6: **PARCUBE outputs sparse factors**: Relative Output size (PARCUBE/ ALS-PARAFAC) vs Relative cost. We see that the results of PARCUBE are more than 90% sparser than the ones from Tensor Toolbox, while maintaining the same approximation error.

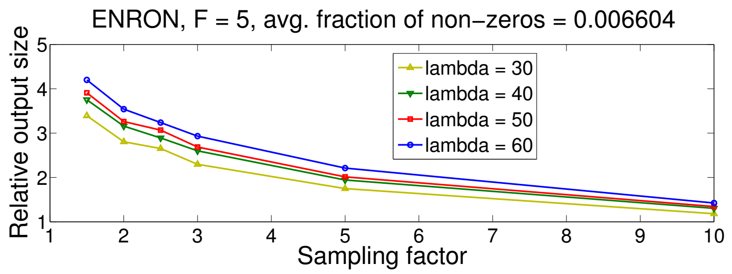


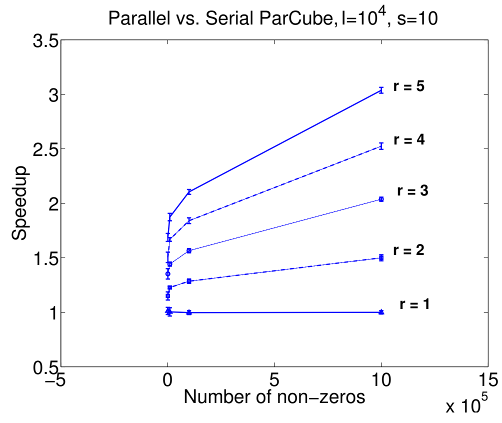
Figure 4.7: **PARCUBE outputs sparse factors**: Relative Output size (PARCUBE/ PARAFAC-SLF) vs sampling factor s (where no. of repetitions is $r = 2s$.

4.3.3 Parallelizability

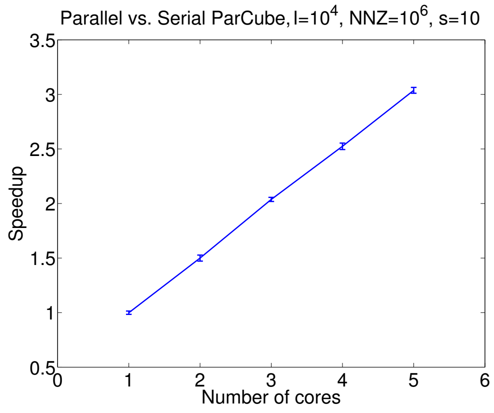
As we discuss earlier, ParCube is inherently a parallel algorithm. Here we investigate the speedup gained through parallelism, as a function of the data size (measured in number of non-zeros) and the number of cores/parallel workers. We set the number of repetitions r to be equal to the number of parallel workers.

Figure 4.8 shows our results: Subfigure 4.8(a) contains the speedup for different number of parallel workers, as the number of non-zeros increases. We observe that when as the number of non-zeros increases, the speedup due to parallelizability becomes more pronounced; intuitively, this result makes sense, since the more dense the original data is, the more dense the samples will be, and therefore, the longer it takes for the Parafac decomposition to be computed for each sample. For this particular test case, we observe a monotonic increase of the speedup as the number of non-zeros increases.

Subfigure 4.8(b) shows the speedup as a function of the parallel workers for a fixed number of non-zeros (the largest one we used for this particular experiment). Here we observe near linear speedup, indicating that the overhead induced by the serial part of the parallel version of ParCube is not a bottleneck, and therefore the parallel version of ParCube scales very well, especially for large input data.



(a)



(b)

Figure 4.8: **ParCube scales well with number of workers**: Serial vs. Parallel ParCube

4.3.4 Scalability in terms of data & parameter size

In addition to measuring ParCube's performance with respect to speeding up a state of the art solver for Parafac, we also measure ParCube's ability to scale in three axes:

1. **Input data size** (measured in number of non-zeros): In Fig. 4.9, we see how ParCube scales as the number of non-zeros grows. We have $r = 4$ repetitions, and equal number of cores, $I = J = K = 10^7$, and the sampling factor is 10^4, essentially resulting in the parallel decomposition of 4 $10^3 \times 10^3 \times 10^3$ tensors. We can see that ParCube scales near-linearly with the number of non-zeros.

2. **Input dimensionality** (measured in the mode sizes of the tensor): In Fig. 4.10, we test how can ParCube scale as the dimensions of the tensor grow. In order to keep other things constant, we keep the number of non-zeros equal to 10^6; as $I = J = K$ grow, this results in increasingly sparser tensors, which from a data analysis point of view might not offer useful information. However, from the viewpoint of testing ParCube's scalability, this experiment provides an insight on how ParCube behaves on very high dimensional tensors. Indeed, ParCube scales near-linearly as the dimensionality of the tensor grows.
3. **Decomposition rank**: Fig. 4.11 demonstrates how ParCube scales as the rank of the decomposition increases. In scenarios where the tensor dimensions are in the orders of 10^7 (as in Fig. 4.11) it is reasonable to seek a decomposition of rank larger than, say, 10 or 20. As the Figure shows, ParCube is able to handle the growth of the rank without experiencing a significant increase in the execution time, thus being scalable in the decomposition rank.

We ran the above experiments 5 independent times, and as the error-bars indicate, the variability of the results is minimal, thus ParCube is consistent in terms of scalability.

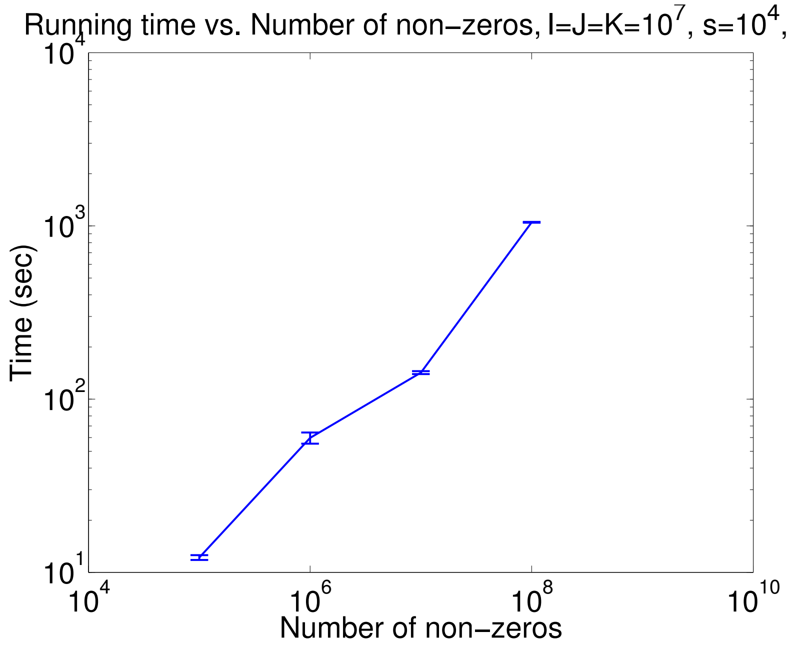


Figure 4.9: Scalability with respect to the number of non-zeros.

4.3.5 Accuracy as a function of tensor density

Here, we experimentally demonstrate that ParCube's performance is consistent for tensors of varying density. In particular, we created a series of randomly generated $10^2 \times 10^2 \times 10^2$ tensors, with varying number of non-zeros, ranging from fully dense (i.e. 10^6 non-zeros), to 0.99 percent sparse (10^4 non-zeros). In order to estimate the stability of

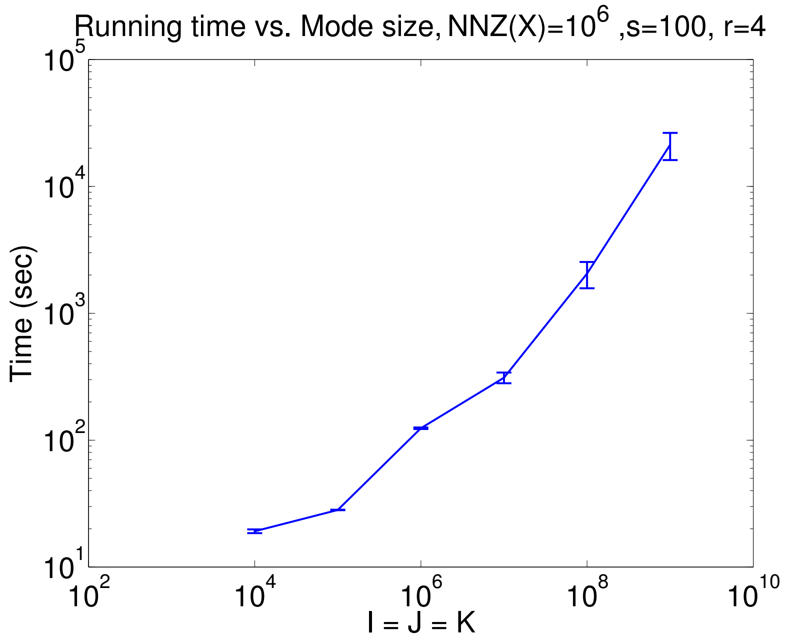


Figure 4.10: Scalability with respect to the tensor dimensions $I = J = K$.

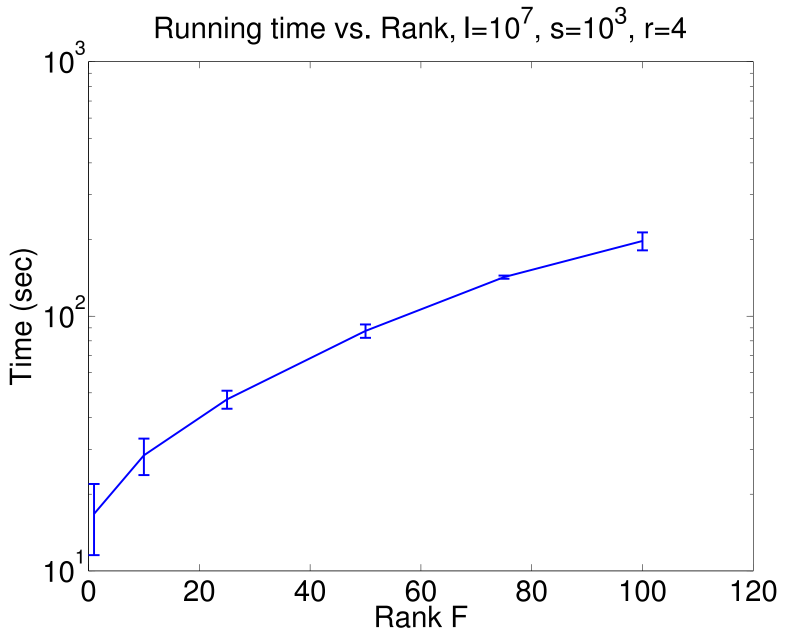


Figure 4.11: Scalability with respect to the decomposition rank.

our results, we ran 50 independent iterations of this experiment, for all different tensors. In Figure 4.12 we present the results of this experiment (where $F = 3$, the sampling factor is $s = 2$ and the number of repetitions were $r = 10$). We observe that the relative cost remains very close to 1 for all different densities that we examined, and the results seem to be very consistent, as indicated by the small error-bars around each point of the figure. Therefore, PARCUBE is able to perform well in a wide range of scenarios, from fully dense tensors, to very sparse ones, as well as for tensors within that spectrum.

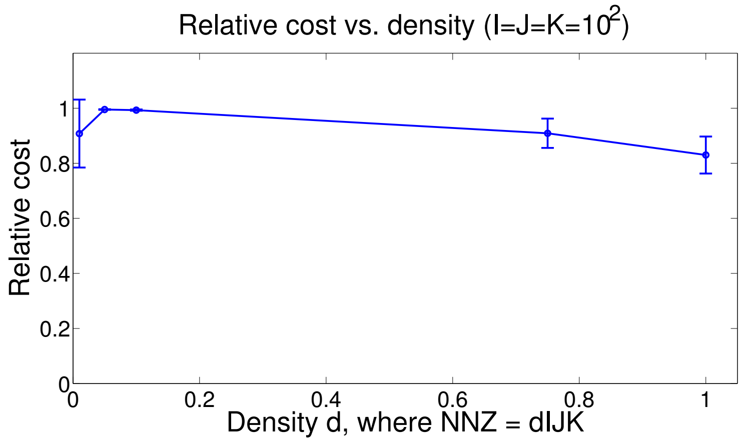


Figure 4.12: Relative errors as a function of the density of the tensor, for $F = 3$, sampling factor $s = 2$ and $r = 10$ repetitions. The density d is defined as $NNZ(\underline{\mathbf{X}}) = dIJK$.

4.3.6 Comparison against TUCKER compression

As we highlighted in Chapter 3, one approach of reducing the size of the tensor into a smaller, compressed version is via the TUCKER decomposition. We compare against the method introduced in [BSG99], where the tensor is first compressed using TUCKER, PARAFAC is fitted in the compressed data and the factor matrices of the TUCKER model are used to decompress the results. In order to compute the TUCKER decomposition, we use the highly optimized Memory Efficient Tucker (MET) algorithm [KS08], included in the Tensor Toolbox for Matlab

In order to ensure a fair comparison, we chose the parameters of both algorithms so that we have the same size for the reduced-size tensor(s). More specifically, we choose $s = 100$ for PARCUBE, and $P = Q = R = 100$ the dimensions of the TUCKER core, while the original tensor is of dimensions $10^4 \times 10^4 \times 10^4$.

Figure 4.13 clearly shows that PARCUBE is orders of magnitude faster than TUCKER compression. The reason why this behavior is observed is because computing the TUCKER decomposition on the full data entails a similar Alternating Least Squares algorithm such as the one used for PARAFAC; therefore, it suffers from similar issues, becoming the bottleneck, even when using a highly optimized algorithm such as MET. On the other hand, the sampling step of PARCUBE is, in practice, much faster than computing the TUCKER decomposition on the full data, and thus PARCUBE ends up being significantly faster.

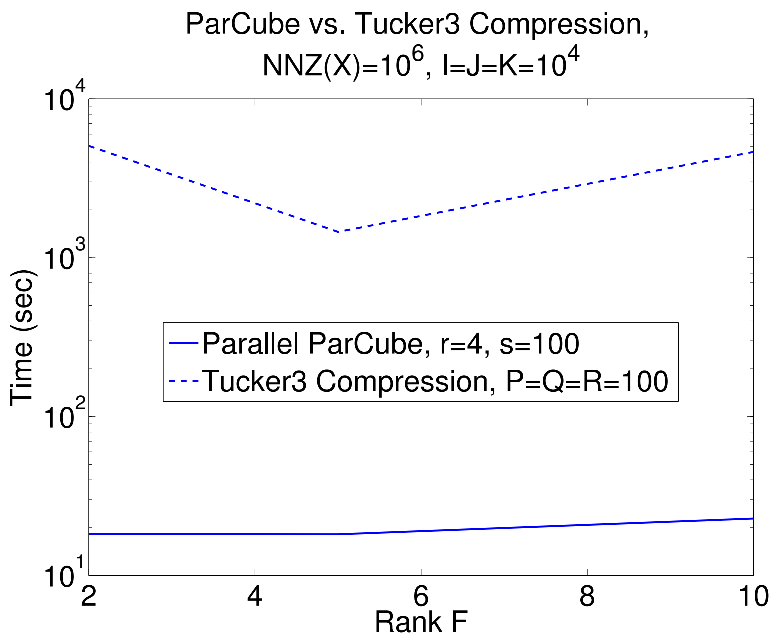


Figure 4.13: **ParCube is orders of magnitude faster than Tucker-based compression**: Parallel ParCube against Tucker compression accelerated Parafac.

4.4 ParCube at work

In this section we present interesting patterns and anomalies, that we were able to discover in the datasets of Table 4.1, demonstrating that our proposed algorithm ParCube is both practical and effective for data mining practitioners. So far, we don't have an automated method for the selection of parameters s, r, and p, but we leave this for future work; the choice is now made empirically.

4.4.1 ENRON

This very well known dataset contains records for 44 months (between 1998 and 2002) of the number of emails exchanged between the 186 employees of the company, forming a $186 \times 186 \times 44$ of 9838 non-zero entries. We executed Algorithm 4.3 using $s = 2$ and $r = 4$ and we applied similar analysis to the resulting factors as the one applied in [BHK06, PSB13]. In Figure 4.14 we illustrate the temporal evolution of the 4 most prevailing groups in our analysis for every month, having annotated the figure with important events, corresponding to peaks in the communication activity. Labelling of the groups was done manually; because the factors were not very sparse we filtered out very low values on each factor. This issue most certainly stems from the fact that this dataset is not particularly large and therefore by applying the regular ALS-Parafac algorithm to the samples (which is known to yield dense factors), we end up with dense sample factors, which eventually, due to repetition, tend to cover most of the data points. This, however, was not the case for larger datasets analyzed in the following lines, for which the factors turned out to be extremely sparse.

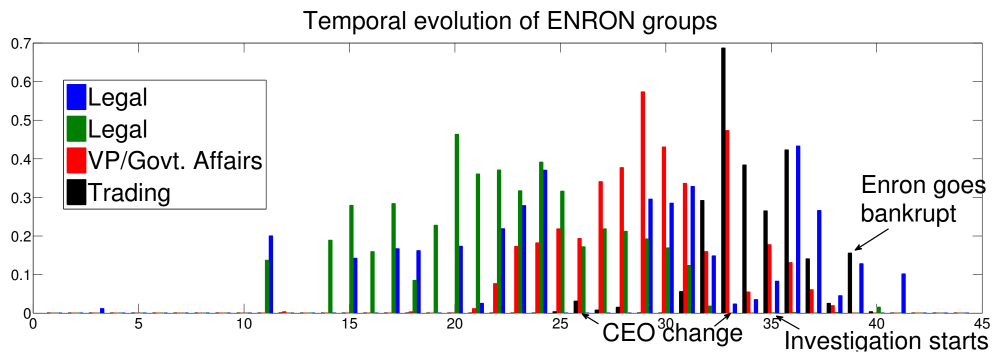


Figure 4.14: **Temporal evolution of 4 groups in the ENRON dataset**: We have labelled the groups, according to the position of the participants in the company. The labels of the extracted groups are consistent with other works in the literature albeit they have been extracted with somewhat different order. We have also discovered 2 *Legal* groups that behave slightly differently over time, a fact probably stemming from the different people involved in each group.

4.4.2 LBNL Network Traffic

This dataset consists of (source, destination, port #) triplets, where each value of the corresponding tensor is the number of packets sent. The snapshot of the dataset we used, formed a $65170 \times 65170 \times 65327$ tensor of 27269 non-zeros. We ran Algorithm 4.3 using $s = 5$ and $r = 10$ and we were able to identify what appears to be a port-scanning attack: The component shown in Fig. 4.15 contains only one source address (addr. 29571), contacting one destination address (addr. 30483) using a wide range of near-consecutive ports (while sending the same amount of packets to each port), a behaviour which should certainly raise a flag to the network administrator, indicating a possible port-scanning attack.

4.4.3 `Facebook` Wall posts

This dataset [3] first appeared in [VMCG09]; the specific part of the dataset we used consists of triplets of the form (Wall owner, Poster, day), where the Poster created a post on the Wall owner's Wall on the specified timestamp. By choosing daily granularity, we formed a $63891 \times 63890 \times 1847$ tensor, comprised of 737778 non-zero entries; subsequently, we ran Algorithm 4.3 using $s = 100$ and $r = 10$. In Figure 4.16 we present our most surprising findings: On the left subfigure, we demonstrate what appears to be the Wall owner's birthday, since many posters posted on a single day on this person's Wall; this event may well be characterized as an "anomaly". On the right subfigure, we demonstrate what "normal" `Facebook` activity looks like.

[3] Download `Facebook` at `http://socialnetworks.mpi-sws.org/data-wosn2009.html`

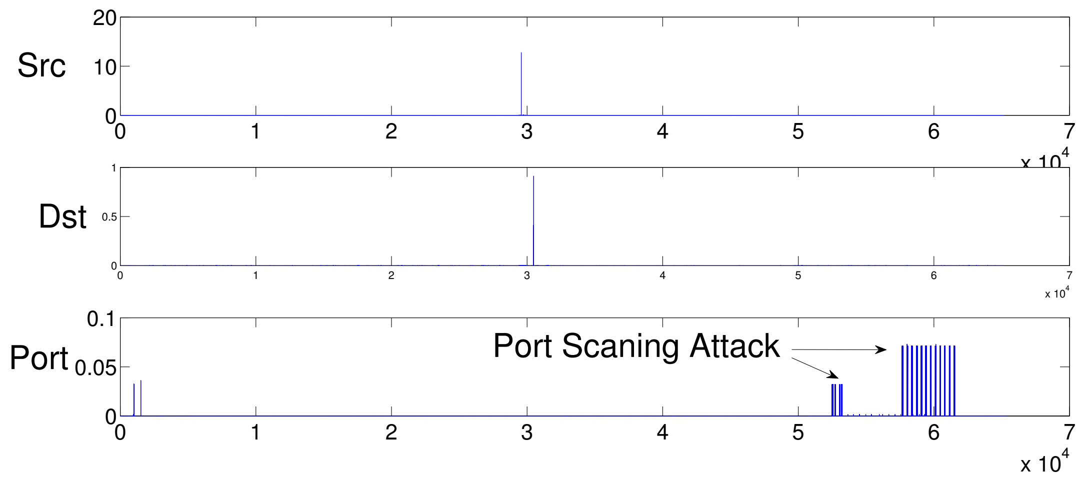


Figure 4.15: **Anomaly on the LBNL data**: We have one source address (addr. 29571), contacting one destination address (addr. 30483) using a wide range of near-consecutive ports, possibly indicating a port scanning attack.

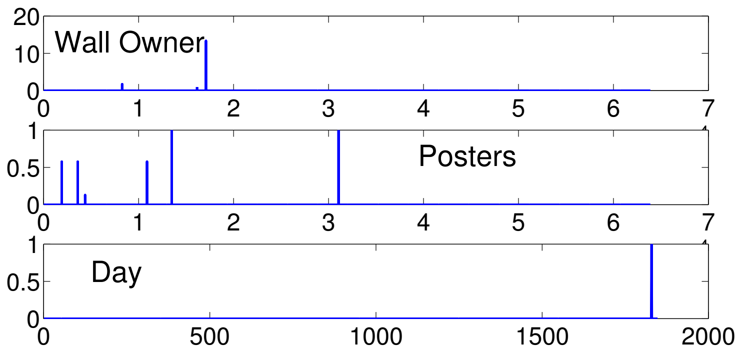
4.4.4 NELL

This dataset consists of triplets of the form (noun-phrase, noun-phrase, context). which form a tensor with assorted modes of size $14545 \times 14545 \times 28818$ and 76879419 non-zeros, and as values the number of occurrences of each triplet. The context phrase may be just a verb or a whole sentence. The PARAFAC decomposition is able to give us latent concepts of noun-phrases that are contextually similar. We used PARCUBE to compute a $F = 50$ component PARAFAC decomposition of this dataset. The sampling factor was set to $s = 50$ and the number of repetitions $r = 20$, and we used 12 workers. Since this dataset is significantly larger than the other three we analyzed, it is worth mentioning that the total running time was 86 minutes. The factors produced were very sparse, with their relative sparsity being:

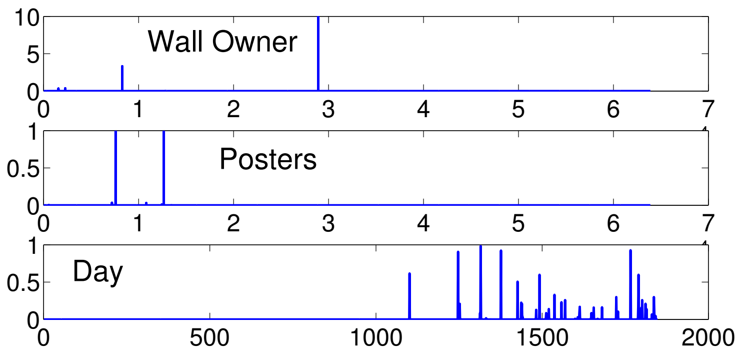
$$\frac{NNZ(\mathbf{A}) + NNZ(\mathbf{B}) + NNZ(\mathbf{C})}{IF + JF + KF} = 0.2$$

We were not able to compute the exact PARAFAC decomposition on a single machine, and thus, we estimate the number of non-zeros of a fully dense matrix $\mathbf{A}$ as IF (and accordingly for the remaining factors).

After computing the PARAFAC decomposition we computed the noun-phrase similarity matrix $\mathbf{A}\mathbf{A}^T + \mathbf{B}\mathbf{B}^T$ and out of that, we were able to discover *contextual* synonyms to noun-phrases, that we report on Table 4.2; the relationship between the words in that table can be viewed as being contextually similar. Additionally, in Table 4.3, we show 10 out of the 50 components that we extracted (in particular, we show the top-3 noun-phrases and context terms). Each row of the Table corresponds to a single concept, and the way to interpret it is the following: The first column shows the top-3 noun-phrases in the first position, the second column contains the second noun-phrase, and the third



(a) Facebook anomaly (Wall owner's birthday)



(b) Facebook normal activity

Figure 4.16: Results for Facebook using $s = 100, r = 10, F = 15$. Subfigure (a): Facebook "anomaly": One Wall, many posters and only one day. This possibly indicates the birthday of the Wall owner. Subfigure(b): Facebook "normal" activity: Many users post on many users' Walls, having a continuous daily activity

column contains the context phrase that connects these two noun-phrases. We observe that the concepts extracted are coherent and meaningful.

Noun-phrase	**Potential Contextual Synonyms**
computer	development
period	day, life
months	life
facilities	families, people, communities
rooms	facilities
legs	people
communities	facilities, families, students
company	community, life, family
groups	people, companies, men
life	experience, day, home
data	information, life, business
people	members, companies, children
countries	people, areas, companies
details	part, information, end
clients	people, children, customers
ability	order, life, opportunity

Table 4.2: NELL: Potential synonym discovery

Noun-phrase 1 (np1)	**Noun-phrase 2 (np2)**	**Context between np1 & np2**
day, time, events	year, month, week	*np1* throughout *np2*, *np1* during *np2*, *np1* last *np2*
information, data, details	site, program, research	*np2* and contact *np1*, *np1* on our *np2*, *np1* provided by *np2*
information, services, data	use, value, variety	*np1* through *np2*, *np2* of their *np1*, *p1* to make *np2*
family, friend, company	support, home, one	*np2* of my *np1*, *np2* of their *np1*, *np2* of her *np1*
areas, countries, communities	services, work, people	*np2* in various *np1*, *np2* within *np1*, *np1* such as *np2*
knowledge, development, needs	students, members, users	*np1* of our *np2*, *np1* of their *np2*, *np1* of his *np2*
business, internet, data	information, system, services	*np1* management *np2*, *np1* software *np2*, *np2* including *np1*
access, changes, information	services, site, students	*np1* to our *np2*, *np1* to my *np2*, *np1* through *np2*
customers, clients, members	quality, value, success	*np2* of their *np1*, *np2* of our *np1*, *np2* of my *np1*
community, country, company	information, services, issue	*np2* within *np1*, *np2* in our *np1*, *np2* across *np1*

Table 4.3: NELL: Concepts of noun-phrases and context words

4.5 Conclusions

In this chapter we have introduced PARCUBE, a novel, fast, parallelizable tensor decomposition which produces sparse factors by construction. Furthermore, it enables processing of large tensors that may not fit in memory. We provide theoretical results that indicate correctness of our algorithm; one of our core contributions pertains to the correct merging of the individual samples. We have demonstrated its merits with respect to sparsity and speedup, compared to the current state of the art, through extensive experimentation. Moreover, we provide a publicly available, highly scalable parallel implementation. Finally, we highlight the practicality of PARCUBE by analyzing four different real datasets, discovering patterns and anomalies.

Chapter based on material published in [PMS+14, PMS+16].

Chapter 5

TURBO-SMT: Parallel Coupled Sparse Matrix-Tensor Factorization

Exploiting sparsity and sampling for paralellizing and speeding up CMTF.

In this Chapter we introduce TURBO-SMT, a tripe-sparse algorithm capable of boosting the performance of any Coupled Matrix-Tensor Factorization (CMTF) algorithm so that it can operate on very large datasets that may not fit in main memory. TURBO-SMT parallelizes *any* CMTF algorithm, producing sparse and interpretable solutions (up to *65 fold*).

5.1 Introduction

How is knowledge mapped and stored in the human brain? How is it expressed by people answering simple questions about specific words? If we have data from both worlds, are we able to combine them and jointly analyze them? In a very different scenario, suppose we have the social network graph of an online social network, and we also have additional information about how and when users interacted with each other. What is a comprehensive way to combine those two pieces of data? Both, seemingly different, problems may be viewed as instances of what is called *Coupled Matrix-Tensor Factorization* (CMTF), where a data tensor and matrices that hold additional information are jointly decomposed into a set of low-rank factors.

The CMTF framework is by no means a new concept, with one of its earliest applications dating back to 2007 where the authors of [BBM07] define a clustering algorithm on data that form Matrix/Tensor couples and apply it to movie recommendation and newsgroup articles settings. Subsequently, there has been a lot of work, both on the algorithmic side [WCVM09, AKD11] and on the application side, ranging from metabolomics [ARS+13] to social network analysis [LSC+09], where usually the data were of small to medium scale. However, we envision applications of the framework where the data scale renders the current state of the art a bottleneck to the analysis. Such an example of application is the one of *Neurosemantics*, where people are shown stimuli (which can range from single words to entire books) and their brain activity is recorded. These measurements can easily span multiple gigabytes of data, and their size and grows as the complexity of the stimulus increases. In fact, in Chapter 9, we will demonstrate a successful application of TURBO-SMT in Neurosemantics. In this Chapter, however, we are concerned with the design of an algorithm for CMTF that is fast, parallelizable, and is able to handle data that may not fit in the main memory of a machine.

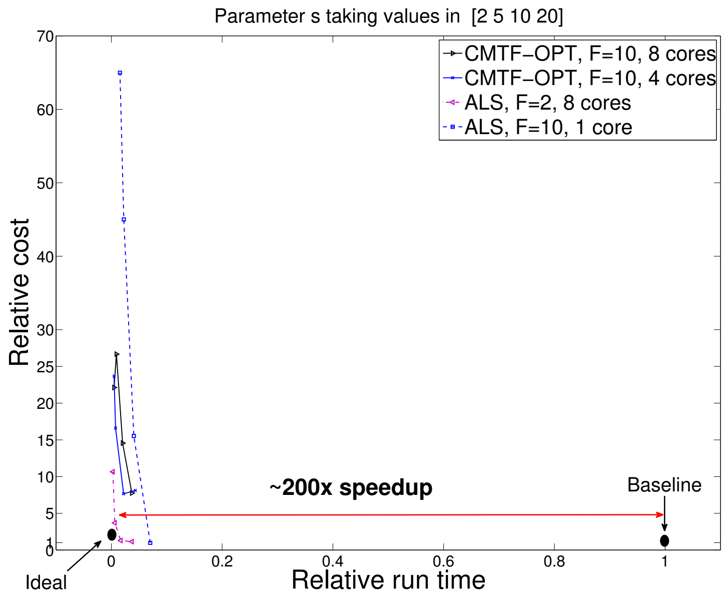


Figure 5.1: TURBO-SMT is up to 200 times faster than standard, state-of-the-art baselines.

In this Chapter, we introduce TURBO-SMT, a parallel, scalable, and sparsity promoting CMTF meta-algorithm. Figure 5.1 shows a snapshot of TURBO-SMT's performance, where it was measured to be up to 200 times faster than standard baselines. In Section 8.4 we present a very detailed experimental evaluation of TURBO-SMT.

Our main contributions are the following:

- *Parallel & triple-sparse algorithm:* We provide an approximate, novel, scalable, and triple-sparse (see Sec. 5.3.1) meta-method, TURBO-SMT, that is able to parallelize and scale-up *any* CMTF core algorithm. Additionally, we improve upon the ALS core CMTF algorithm, making it faster and able to handle missing values (see Sec. 5.3.1 for details on the algorithm, and Sec. 5.5 for experimental evaluation).

- *Reproducibility:* Our code is publicly available [1].

5.2 Preliminaries

5.2.1 A note on notation

The notation used here follows Chapter 2 but we augment it with a few symbols used particularly in this Chapter. Table 5.1 contains symbols and essential notation that is henceforth used throughout the text.

Symbol	Description
CMTF	Coupled Matrix-Tensor Factorization
ALS	Alternating Least Squares
CMTF-OPT	Algorithm introduced in [AKD11]
$x, \mathbf{x}, \mathbf{X}, \underline{\mathbf{X}}$	scalar, vector, matrix, tensor (respectively)
$\mathbf{A} \odot \mathbf{B}$	Khatri-rao product.
$\mathbf{A} \otimes \mathbf{B}$	Kronecker product.
$\mathbf{A} * \mathbf{B}$	Hadamard (elementwise) product.
$\mathbf{A}^{\dagger}$	Pseudoinverse of $\mathbf{A}$
$\|\mathbf{A}\|_F$	Frobenius norm of $\mathbf{A}$.
$\mathbf{a} \circ \mathbf{b} \circ \mathbf{c}$	$(\mathbf{a} \circ \mathbf{b} \circ \mathbf{c})\,(i,j,k) = \mathbf{a}(i)\mathbf{b}(j)\mathbf{c}(k)$
(i) as superscript	Indicates the i-th iteration
$\mathbf{A}_1^i$, $\mathbf{a}_1^i$	series of matrices or vectors, indexed by i.
$\underline{\mathbf{X}}_{(i)}$	i-th mode unfolding of tensor $\underline{\mathbf{X}}$ (see [Kie00]).
$\mathcal{I}$	Set of indices.
$\mathbf{x}(\mathcal{I})$	Spanning indices $\mathcal{I}$ of $\mathbf{x}$.
BIASEDSAMPLE	a biased random sample of the data tensor.

Table 5.1: Table of symbols

5.2.2 Coupled Matrix-Tensor Factorization

As we saw in Chapter 2, oftentimes, two tensors, or a matrix and a tensor, may have one mode in common; consider the example that we mentioned earlier, where we have a word by brain activity by human subject tensor, we also have a semantic matrix that provides additional information for the same set of words. In this case, we say that the matrix and the tensor are *coupled* in the 'word' mode, and the problem is instance of Coupled Matrix-Tensor Factorization (CMTF) [BBM07, WCVM09, AKD11] which is an active field of research, aiming to jointly analyze datasets (in the form of tensors and matrices) that share a subset of their modes.

In this work we focus on three mode tensors, however, everything we mention extends directly to higher modes. In the general case, a three mode tensor $\underline{\mathbf{X}}$ may be coupled with at most three matrices $\mathbf{Y}_i,\ i = 1 \cdots 3$, in the manner illustrated in Figure 5.2 for one

[1] www.cs.cmu.edu/~epapalex/src/turbo_smt.zip

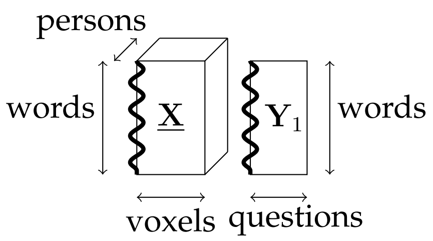


Figure 5.2: *Coupled Matrix - Tensor* example: Tensors often share one or more modes (with thick, wavy line): $\underline{\mathbf{X}}$ is the brain activity tensor and $\mathbf{Y}$ is the semantic matrix. As the wavy line indicates, these two datasets are coupled in the "word" dimension.

mode. The optimization function that encodes this decomposition is:

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{D},\mathbf{E},\mathbf{G}} \|\underline{\mathbf{X}} - \sum_k \mathbf{a}_k \circ \mathbf{b}_k \circ \mathbf{c}_k\|_F^2 + \\ \|\mathbf{Y}_1 - \mathbf{A}\mathbf{D}^T\|_F^2 + \|\mathbf{Y}_2 - \mathbf{B}\mathbf{E}^T\|_F^2 + \|\mathbf{Y}_3 - \mathbf{C}\mathbf{G}^T\|_F^2 \tag{5.1}$$

where $\mathbf{a}_k$ is the k-th column of $\mathbf{A}$. The idea behind the coupled matrix-tensor decomposition is that we seek to jointly analyze $\underline{\mathbf{X}}$ and $\mathbf{Y}_i$, decomposing them to latent factors who are coupled in the shared dimension. For instance, the first mode of $\underline{\mathbf{X}}$ shares the same low rank column subspace as $\mathbf{Y}_1$; this is expressed through the latent factor matrix $\mathbf{A}$ which jointly provides a basis for that subspace.

For more details on algorithms for CMTF, we refer the reader to Chapter 3.

5.3 Proposed Method

5.3.1 Algorithm Description

There are three main concepts behind TURBO-SMT (outlined in Algorithm 5.1):

Phase 1 Obtain a sample of our data by using biased sampling.
Phase 2 Fit CMTF to the reduced data (possibly on more than one samples)
Phase 3 stitch the partial results

Phase1: Sampling An efficient way to reduce the size of the dataset, yet operate on a representative subset thereof is to use *biased* sampling. In particular, given a three-mode tensor $\underline{\mathbf{X}}$ we sample as follows. We calculate three vectors as shown in equations (5.2), one for each mode of $\underline{\mathbf{X}}$ (and respectively Equations 5.3 and 5.4 for the two modes of the matrix).

These vectors, which we henceforth refer to as *density vectors* are the marginal absolute sums with respect to all but one of the modes of the tensor, and in essence represent the importance of each index of the respective mode. We then sample *indices* of each mode according to the respective density vector. For instance, assume an $I \times J \times K$ tensor; suppose that we need a sample of size $\frac{I}{s}$ of the indices of the first mode. Then, we just define $p_{\mathcal{I}}(i) = \mathbf{x}_A(i) / \sum_{i=1}^{I} \mathbf{x}_A(i)$ as the probability of sampling the i-th index of the first

$$\mathbf{x}_A(i) = \sum_{j=1}^{J}\sum_{k=1}^{K} |\underline{\mathbf{X}}(i,j,k)| + \sum_{j=1}^{I_1} |\mathbf{Y}_1(i,j)|, \quad \mathbf{x}_B(j) = \sum_{i=1}^{I}\sum_{k=1}^{K} |\underline{\mathbf{X}}(i,j,k)| + \sum_{i=1}^{I_2} |\mathbf{Y}_2(j,i)|, \quad \mathbf{x}_C(k) = \sum_{i=1}^{I}\sum_{j=1}^{J} |\underline{\mathbf{X}}(i,j,k)| + \sum_{j=1}^{I_3} |\mathbf{Y}_3(k,j)|, \tag{5.2}$$

$$\mathbf{y}_{1,A}(i) = \sum_{j=1}^{I_1} |\mathbf{Y}_1(i,j)| \quad \mathbf{y}_{2,B}(j) = \sum_{i=1}^{I_2} |\mathbf{Y}_2(j,i)|, \quad \mathbf{y}_{3,C}(k) = \sum_{j=1}^{I_3} |\mathbf{Y}_3(k,j)| \tag{5.3}$$

$$\mathbf{y}_{1,D}(j) = \sum_{i=1}^{I} |\mathbf{Y}_1(i,j)|, \quad \mathbf{y}_{2,G}(i) = \sum_{j=1}^{J} |\mathbf{Y}_2(j,i)|, \quad \mathbf{y}_{3,E}(i) = \sum_{k=1}^{K} |\mathbf{Y}_3(k,i)| \tag{5.4}$$

mode, and we simply sample without replacement from the set $\{1 \cdots I\}$, using $p_{\mathcal{I}}$ as bias. The very same idea is used for matrices $\mathbf{Y}_i$. Doing so is preferable over sampling uniformly, since our bias makes it more probable that high density indices of the data will be retained on the sample, and hence, it will be more representative of the entire set.

Suppose that we call $\mathcal{I}, \mathcal{J}, \mathcal{K}$ the index samples for the three modes of $\underline{\mathbf{X}}$. Then, we may take $\underline{\mathbf{X}}_s = \underline{\mathbf{X}}(\mathcal{I}, \mathcal{J}, \mathcal{K})$ (and similarly for matrices $\mathbf{Y}_i$) to form a sample of the data; which essentially is a small, yet representative, sample of our original dataset, where the high density blocks are more likely to appear on the sample. It is important to note that the indices of the coupled modes are the same for the matrix and the tensor, e.g. $\mathbf{I}$ randomly selects the same set of indices for $\underline{\mathbf{X}}$ and $\mathbf{Y}_1$. This way, we make sure that the coupling is *preserved* after sampling.

In cases where the data have very different dimensions per mode (e.g. one mode is significantly smaller than the rest), we may use different sampling factors in order to account for this imbalance.

Finally, Phase 1 can be executed very efficiently, since both the calculation of sample biases, as well as the sampling of indices require only 2 passes on the non-zero elements of the (usually, highly sparse) data.

Phase 2: Fit CMTF to samples The next step of TURBO-SMT is to fit a CMTF model to each sample, and then, based on the sampled indices, redistribute the result to the original index space. As we have already discussed, TURBO-SMT is not restricted in any way to a specific CMTF solver; in fact, we provide experiments using both an ALS and a first order gradient approach. In more detail, suppose that $\mathbf{A}_s$ is the factor matrix obtained by the aforementioned procedure, and that jointly describes the first mode of $\underline{\mathbf{X}}_s$ and $\mathbf{Y}_{1,s}$. The dimensions of $\mathbf{A}_s$ are going to be $|\mathcal{I}| \times F$ (where $||$ denotes cardinality and F is the number of factors). Let us further assume matrix $\mathbf{A}$ of size $I \times F$ which expresses the first mode of the tensor and the matrix, before sampling; due to sampling, it holds that $I \gg |\mathcal{I}|$. If we initially set all entries of $\mathbf{A}$ to zero and we further set $\mathbf{A}(\mathcal{I}, :) = \mathbf{A}_s$ we obtain a highly *sparse* factor matrix whose non-zero values are a 'best effort' approximation of the true ones, i.e. the values of the factor matrix that we would obtain by decomposing the full data.

So far, we have provided a description of the algorithm where only one repetition of sampling is used. However, the approximation quality of TURBO-SMT improves as we increase the number of repetitions. To that end, we allow for multiple sampling repetitions in our algorithm, i.e. extracting multiple sample tensors $\underline{\mathbf{X}}_s$ and side matrices $\mathbf{Y}_{i,s}$, fitting a CMTF model to all of them and combining the results in a way that the true latent patterns are retained. We are going to provide a detailed outline of how to carry the multi-repetition version of TURBO-SMT in the following.

While doing multiple repetitions, we keep a *common* subset of indices for all different samples. In particular, let p be the percentage of common values across all repetitions and $\mathcal{I}_p$ denote the common indices along the first mode (same notation applies to the rest of the indices); then, all sample tensors $\underline{\mathbf{X}}_s$ will definitely contain the indices $\mathcal{I}_p$ on the first mode, as well as $(1-p)\frac{I}{s}$ indices sampled independently (across repetitions) at random. This common index sample is key in order to ensure that our results are not rank deficient, and all partial results are merged correctly. As we discuss in Phase 3, it suffices to keep a set of common "anchor" indices just for first mode, however, we also describe a method that uses common in all modes of the tensor (i,e, $\mathcal{I}_p, \mathcal{J}_p, \mathcal{K}_p$), that is, however, more expensive computationally.

We do not provide an exact method for choosing p, however, as a rule of thumb, we observed that, depending on how sparse and noisy the data is, a range of p between 0.2 and 0.5 works well. This introduces a trade-off between redundancy of indices that we sample, versus the accuracy of the decomposition; since we are not dealing solely with tensors, which are known to be relatively more well behaved in terms of decomposition uniqueness (in contrast to matrices), it pays off to introduce some data redundancy (especially when TURBO-SMT runs in a parallel system) so that we avoid rank-deficiency in our data.

Let r be the number of different sampling repetitions, resulting in r different sets of sampled matrix-tensor couples $\underline{\mathbf{X}}_s^{(i)}$ and $\mathbf{Y}_{j,s}^{(i)}$ ($i = 1 \cdots r,\ j = 1 \cdots 3$). For that set of coupled data, we fit a CMTF model, using a CMTF solver, obtaining a set of factor matrices $\mathbf{A}^{(i)}$ (and likewise for the rest).

Phase 3: Stitching partial results After having obtained these r different sets of partial results, as a final step, we have to merge them together into a set of factor matrices that we would ideally get had we operated on the full dataset.

In order to make the merging work, we first introduce the following scaling on each column of each factor matrix: Let's take $\mathbf{A}^{(i)}$ for example; we normalize each column of $\mathbf{A}$ by the ℓ_2 norm of the common part, as described in line 8 of Algorithm 5.1. By doing so, the common part of each factor matrix (for all repetitions) will be unit norm. This scaling is absorbed in a set of scaling vectors $\boldsymbol{\lambda}_A$ (and accordingly for the rest of the factors). The new objective function is shown in Equation 5.5

$$\begin{aligned}\min_{\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{D},\mathbf{E},\mathbf{G}} &\|\underline{\mathbf{X}} - \sum_k \boldsymbol{\lambda}_A(k)\boldsymbol{\lambda}_B(k)\boldsymbol{\lambda}_C(k)\mathbf{a}_k \circ \mathbf{b}_k \circ \mathbf{c}_k\|_F^2 \\ &+ \|\mathbf{Y}_1 - \mathbf{A}\ \mathrm{diag}(\boldsymbol{\lambda}_A * \boldsymbol{\lambda}_D)\ \mathbf{D}^T\|_F^2 \\ &+ \|\mathbf{Y}_2 - \mathbf{B}\ \mathrm{diag}(\boldsymbol{\lambda}_B * \boldsymbol{\lambda}_E)\ \mathbf{E}^T\|_F^2 \\ &+ \|\mathbf{Y}_3 - \mathbf{C}\ \mathrm{diag}(\boldsymbol{\lambda}_C * \boldsymbol{\lambda}_G)\ \mathbf{G}^T\|_F^2\end{aligned} \tag{5.5}$$

A problem that is introduced by carrying out multiple sampling repetitions is that the correspondence of the output factors of each repetition is very likely to be distorted. In other words, say we have matrices $\mathbf{A}^{(1)}$ and $\mathbf{A}^{(2)}$ and we wish to merge their columns (i.e. the latent components) into a single matrix $\mathbf{A}$, by stitching together columns that correspond to the same component. It might very well be the case that the order in which the latent components appear in $\mathbf{A}^{(1)}$ is not the same as in $\mathbf{A}^{(2)}$.

The sole purpose of the aforementioned normalization is to resolve the correspondence problem. In Algorithm 5.2, we merge the partial results while establishing the correct correspondence of the columns.

Algorithm 5.2 can be seen as a greedy algorithm for solving the correspondence problem between columns. Provided that the factor matrices are not collinear, the algorithm usually finds the correct correspondence. In reality, where data might be noisy, we can use the Hungarian Method [Kuh55], which will slightly increase the computational cost of the overall algorithm but solve the problem optimally.

In order to see why this happens, we follow the example of $r = 2$ of the previous paragraph, according to Algorithm 5.2, we compute the inner product of the common parts of each column of $\mathbf{A}^{(1)}$ and $\mathbf{A}^{(2)}$. Since the common parts of each column are normalized to unit norm, then the inner product of the common part of the column of $\mathbf{A}^{(1)}$ with that of $\mathbf{A}^{(2)}$ will be maximized (and exactly equal to 1) for the matching columns, and by the Cauchy-Schwartz inequality, for all other combinations, it will be less than 1. Additionally, elimination of the already used columns operates as a tie-breaker.

The non-collinearity assumption implies that the matrix/tensor pair of each sample contains data that fit the CMTF model (for the given decomposition rank) well. If either the data do not obey the model, or the rank is too high, the partial factors to be merge might end up being collinear, in which case we have to either decrease the decomposition rank, discard the particular sample that produces collinear factors, or readjust s and r so that the sample chosen has "good" structure (where "good" is used in the sense that it fits the CMTF model well).

Note that in the way that we describe the stitching, we only need to keep common "anchor" indices for the $\mathbf{A}$ matrix, establish the correspondence on $\mathbf{A}$ and then propagate the column permutation to the remaining factor matrices. Since TURBO-SMT can be used for data whose dimensions span millions or billions, a set of anchor indices can easily span thousands or even tens of thousands, thus, saving only a set of anchor indices for matrix $\mathbf{A}$ results in memory and communication savings. Phase 3, due to its low complexity, can be executed very efficiently. In particular, the FACTORMERGE algorithm requires $O(rF^2)$ steps, where, both r and F are, for most practical cases, very small, compared to the data dimensionality.

For the sake of completeness, here we also describe a theoretically more accurate, but more computationally and memory intensive way to stitch the components. Suppose we have a set of common indices for all $\mathbf{A}, \mathbf{B}, \mathbf{C}$. We may, thus, perform the stitching jointly for $\mathbf{A}, \mathbf{B}, \mathbf{C}$ and then propagate the permutation that we computed to matrices $\mathbf{D}, \mathbf{E}, \mathbf{G}$. This joint matching can be done as follows: Consider the following product:

$$\mathbf{A}(\mathcal{I}_p, :) \odot \mathbf{B}(\mathcal{J}_p, :) \odot \mathbf{C}(\mathcal{K}_p, :),$$

where $\odot$ is the Khatri-Rao product, as define earlier. This results in a matrix with F columns, where the f-th column is equal to

$$\mathbf{A}(\mathcal{I}_p, f) \otimes \mathbf{B}(\mathcal{J}_p, f) \otimes \mathbf{C}(\mathcal{K}_p, f).,$$

where $\otimes$ is the Kronecker product. Because of the definition of the PARAFAC decomposition, the above is simply the vectorized f-th rank one component of the PARAFAC decomposition defined by $\mathbf{A}(\mathcal{I}_p, :), \mathbf{B}(\mathcal{J}_p, :), \mathbf{C}(\mathcal{K}_p, :)$, i.e. the pieces of the partial results that correspond to the same "anchor" indices. Thus, the problem of finding the correspondence between, say, the first sample and the second sample reduces to the problem of finding the optimal column correspondence between matrices

$$\mathbf{A}^{(1)}(\mathcal{I}_p, :) \odot \mathbf{B}^{(1)}(\mathcal{J}_p, :) \odot \mathbf{C}^{(1)}(\mathcal{K}_p, :)$$

and

$$\mathbf{A}^{(2)}(\mathcal{I}_p, :) \odot \mathbf{B}^{(2)}(\mathcal{J}_p, :) \odot \mathbf{C}^{(2)}(\mathcal{K}_p, :)$$

which can, again, be solved using the Hungarian Method [Kuh55]. However, as we mentioned earlier, the size of $\mathcal{I}_p, \mathcal{J}_p, \mathcal{K}_p$, in cases where we are dealing with big tensors and matrices, can be in the orders of thousands or tens of thousands. Thus, computing the above Khatri-Rao products may incur high computational and memory overhead during the stitching step, thus, in Algorithm 5.1 we show TURBO-SMT when using the lightweight version of stitching (Algorithm 5.2).

5.3.2 Sparsity through Sampling

Besides data size reduction, one merit of sampling is sparsity on the latent factors. Every time TURBO-SMT does one repetition, it operates on a sub-sampled version of the data. Consequently, in the third phase of the algorithm, where the results are re-distributed to their indices in the original, high dimensional space, most of the indices of the latent factors are going to be exactly zero, thus resulting in latent factor sparsity. In this way, TURBO-SMT always operates on a sparse set of data, through the entire lifetime of the

Algorithm 5.1: TURBO-SMT: Sparse and parallel CMTF

```
Input: Tensor $\underline{\mathbf{X}}$ of size $I \times J \times K$, matrices $\mathbf{Y}_i,\ i = 1 \cdots 3$, of size $I \times I_2$, $J \times J_2$, and $K \times K_2$
    respectively, number of factors $F$, sampling factor $s$, number of repetitions $r$.
Output: $\mathbf{A}$ of size $I \times F$, $\mathbf{b}$ of size $J \times F$, $\mathbf{c}$ of size $K \times F$, $\mathbf{D}$ of size $I_2 \times F$, $\mathbf{G}$ of size $J_2 \times F$, $\mathbf{E}$ of
    size $K_2 \times F$. $\boldsymbol{\lambda}_A$, $\boldsymbol{\lambda}_B$, $\boldsymbol{\lambda}_C$, $\boldsymbol{\lambda}_D$, $\boldsymbol{\lambda}_E$, $\boldsymbol{\lambda}_G$ of size $F \times 1$.
 1: Initialize $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}, \mathbf{E}, \mathbf{G}$ to all-zeros.
 2: Randomly, *using mode densities as bias*, select a set of $100p\%$ ($p \in [0,1]$) "anchor" indices $\mathcal{I}_p$ to
    be common across all repetitions. For example, $\mathcal{I}_p$ is sampled with probabilities with
    $p_{\mathcal{I}}(i) = \mathbf{x}_A(i) / \sum_{i=1}^{I} \mathbf{x}_A(i)$.
 3: for $i = 1 \cdots r$ do
      {Phase 1: Obtain samples through biased sampling}
 4:   Compute densities as in equations 5.2, 5.3, 5.4.
      Compute set of indices $\mathcal{I}^{(i)}$ as random sample without replacement of $\{1 \cdots I\}$ of size
      $I/(s(1-p))$ with probability $p_{\mathcal{I}}(i) = \mathbf{x}_A(i) / \sum_{i=1}^{I} \mathbf{x}_a(i)$. Likewise for $\mathcal{J}, \mathcal{K},, \mathcal{I}_1, \mathcal{I}_2$, and $\mathcal{I}_3$.
      Set $\mathcal{I}^{(i)} = \mathcal{I} \cup \mathcal{I}_p$.
 5:   Get sample $\underline{\mathbf{X}}_s^{(i)} = \underline{\mathbf{X}}(\mathcal{I}^{(i)}, \mathcal{J}^{(i)}, \mathcal{K}^{(i)})$, $\mathbf{Y}_{1s}^{(i)} = \mathbf{Y}_1(\mathcal{I}^{(i)}, \mathcal{I}_1^{(i)})$ and likewise for $\mathbf{Y}_{2s}^{(i)}$ and $\mathbf{Y}_{3s}^{(i)}$.
      Note that the same index sample is used for *coupled* modes.
      {Phase 2: Fit the model on each sample}
 6:   Run a CMTF solver for $\underline{\mathbf{X}}_s^{(i)}$ and $\mathbf{Y}_{js}^{(i)},\ j = 1 \cdots 3$ and obtain $\mathbf{A}_s, \mathbf{B}_s, \mathbf{C}_s, \mathbf{D}_s, \mathbf{G}_s, \mathbf{E}_s$.
 7:   $\mathbf{A}^{(i)}(\mathcal{I}^{(i)}, :) = \mathbf{A}_s$. Likewise for the rest.
 8:   Calculate the $\ell_2$ norm of the columns of the common part: $\boldsymbol{\lambda}_A^{(i)}(f) = \|\mathbf{A}^{(i)}(\mathcal{I}_p, f)\|_2$, for
      $f = 1 \cdots F$. Normalize columns of $\mathbf{A}^{(i)}$ using $\boldsymbol{\lambda}_A^{(i)}$ (likewise for the rest). Note that the
      common part of each factor will now be normalized to unit norm.
 9: end for
    {Phase 3: Stitch partial results}
10: $\mathbf{A}$ =FACTORMERGE ($\mathbf{A}_1^i$). When stitching partial results for $\mathbf{A}$, record the column
    correspondences for different nuclei
11: Using the above correspondence, stitch the partial factors of $\mathbf{B}, \mathbf{C}, \mathbf{D}, \mathbf{E}, \mathbf{G}$, without
    performing the full FACTORMERGE algorithm, but simply reordering the columns.
12: Using the above column/component correspondence, reorder $\boldsymbol{\lambda}_A, \boldsymbol{\lambda}_B, \boldsymbol{\lambda}_C, \boldsymbol{\lambda}_D, \boldsymbol{\lambda}_E, \boldsymbol{\lambda}_G$ for
    each repetition.
13: $\boldsymbol{\lambda}_A$ = average of $\boldsymbol{\lambda}_{A_1^i}$. Likewise for the rest.
```

algorithm, a thing which is not true for the majority of the algorithms both for tensor and coupled decompositions, which usually operate on dense factors (even when the final output is sparse), and have very high and unnecessary storage needs.

5.3.3 Parallelization

TURBO-SMT is, by its nature, parallelizable; in essence, we generate multiple samples of the coupled data, we fit a CMTF model to each sample and then we merge the results. By

Algorithm 5.2: FACTORMERGE: Given partial results of factor matrices, merge them correctly

```
Input: Factor matrices $\mathbf{A}_1^i$ of size $I \times F$ each, and $r$ is the number of repetitions, $\mathcal{I}_p$: set of
    common indices.
Output: Factor matrix $\mathbf{A}$ of size $I \times F$.
 1: Set $\mathbf{A} = \mathbf{A}^{(1)}$
 2: Set $\ell = \{1 \cdots F\}$, a list that keeps track of which columns have not been assigned yet.
 3: for $i = 2 \cdots r$ do
 4:     for $f_1 = 1 \cdots F$ do
 5:         for $f_2$ in $\ell$ do
 6:             Compute similarity $\mathbf{v}(f_2) = \left(\mathbf{A}(\mathcal{I}_p, f_2)\right)^T \left(\mathbf{A}^{(i)}(\mathcal{I}_p, f_1)\right)$
 7:         end for
 8:         $c^* = \arg\max_c \mathbf{v}(c)$ (Ideally, for the matching columns, the inner product should be
            equal to 1; conversely, for the rest of the columns, it should be considerably smaller)
 9:         $\mathbf{A}(:, c^*) = \mathbf{A}^{(i)}(:, f_1)\big|_{\mathbf{A}(:,c^*)=0}$, i.e. update the zero entries of the column.
10:         Remove $c^*$ from list $\ell$.
11:     end for
12: end for
```

carefully observing Algorithm 5.1, we can see that lines 3 to 9 may be carried out entirely in parallel, provided that we have a good enough random number generator that does not generate the very same sample across all r repetitions. In particular, the r repetitions are independent from one another, since computing the set of common indices (line 2), which is the common factor across all repetitions, is done before line 3.

5.4 Further Optimizations

5.4.1 Speeding up the ALS algorithm

In addition to our main contribution in terms of speeding CMTF in general, we are able to further speed the ALS algorithm up, by making a few careful interventions to the core algorithm (Algorithm 3.7).

Lemma 5.1.:
We may do the following simplification to each pseudoinversion step of the ALS algorithm (Algorithm 3.7):

$$\begin{bmatrix} \mathbf{A} \odot \mathbf{B} \\ \mathbf{M} \end{bmatrix}^{\dagger} = \left(\mathbf{A}^T\mathbf{A} * \mathbf{B}^T\mathbf{B} + \mathbf{M}^T * \mathbf{M}\right)^{\dagger} \left[\left(\mathbf{A} \odot \mathbf{B}\right)^T, \mathbf{M}^T\right]$$

Proof. For the Moore-Penrose pseudoinverse of the Khatri-Rao product, it holds that [Bro98, LT08]

$$\left(\mathbf{A} \odot \mathbf{B}\right)^{\dagger} = \left(\mathbf{A}^T\mathbf{A} * \mathbf{B}^T\mathbf{B}\right)^{\dagger} \left(\mathbf{A} \odot \mathbf{B}\right)^T$$

Furthermore [Bro98] $\left(\mathbf{A} \odot \mathbf{B}\right)^T \left(\mathbf{A} \odot \mathbf{B}\right) = \mathbf{A}^T\mathbf{A} * \mathbf{B}^T\mathbf{B}$ For a partitioned matrix $\mathbf{P} = \begin{bmatrix} \mathbf{P}_1 \\ \mathbf{P}_2 \end{bmatrix}$, it holds that its pseudoinverse may be written in the following form [HM75]

$$\begin{bmatrix}\mathbf{P}_1\\ \mathbf{P}_2\end{bmatrix}^{\dagger} = \left(\mathbf{P}_1^T\mathbf{P}_1 + \mathbf{P}_2^T\mathbf{P}_2\right)^{\dagger}\left[\mathbf{P}_1^T, \quad \mathbf{P}_2^T\right]$$

Putting things together, it follows:

$$\begin{bmatrix}\mathbf{A}\odot\mathbf{B}\\ \mathbf{M}\end{bmatrix}^{\dagger} = \left(\mathbf{A}^T\mathbf{A} * \mathbf{B}^T\mathbf{B} + \mathbf{M}^T * \mathbf{M}\right)^{\dagger}\left[\left(\mathbf{A}\odot\mathbf{B}\right)^T, \mathbf{M}^T\right]$$

which concludes the proof. ■

The above lemma implies that substituting the naive pseudoinversion of $\begin{bmatrix}\mathbf{A}\odot\mathbf{B}\\ \mathbf{M}\end{bmatrix}$ with the simplified version, offers significant *computational* gains to Algorithm 3.7. More precisely, if the dimensions of $\mathbf{A}, \mathbf{B}$ and $\mathbf{M}$ are $I\times R$, $J\times R$ and $I\times I_2$, then computing the pseudoinverse naively would cost $O\left(R^2\left(IJ+I_2\right)\right)$, whereas our proposed method yields a cost of $O\left(R^2\left(I+J+I_2\right)\right)$ because of the fact that we are pseudoinverting only a *small* $R\times R$ matrix. We have to note here that in almost all practical scenarios $R \ll I, J, I_2$.

Table 5.2 provides a solid impression of the speedup achieved on the core ALS algorithm, as a result of the simplification of the pseudo-inversion step, as derived above. In short, we can see that the speedup achieved is in most realistic cases 2x or higher, adding up to being a significant improvement on the traditional algorithm.

R	$I = 10$	$I = 100$	$I = 1000$	$I = 10000$	$I = 100000$
1	2.4686 $\pm$ 0.3304	2.4682 $\pm$0.3560	2.4479 $\pm$ 0.2948	2.4546 $\pm$ 0.3214	2.4345 $\pm$ 0.3144
5	2.2496 $\pm$ 0.3134	2.2937 $\pm$ 0.1291	2.2935 $\pm$ 0.1295	2.2953 $\pm$ 0.1291	2.2975 $\pm$ 0.1318
10	2.6614 $\pm$ 0.1346	2.6616 $\pm$ 0.1368	2.6610 $\pm$ 0.1380	2.6591 $\pm$ 0.1377	2.6593 $\pm$ 0.1428

Table 5.2: Pseudoinversion speedup (100000 runs)

5.4.2 Making ALS robust to missing values

In many practical scenarios, we often have corrupted or missing data. For instance, when measuring brain activity, a few sensors might stop working, whereas the majority of the sensors produce useful signal. Despite these common data imperfections, it is important for a data mining algorithm to be able to operate. The work of Tomasi et. al [TB05] provides a very comprehensive study on how to handle missing values for plain tensor decompositions. A very clean and straightforward way of handling missing values is to *ignore* them throughout the optimization process, both with respect to the original data and with respect to the model. This can be achieved through masking the missing values, hiding them from the optimization. Notice that is *not* the same as simply zeroing out all missing values, since 0 might have a valid physical interpretation.

In this section, we show how this masking can be applied to the ALS algorithm for the CMTF model. In [AKD11], the authors show how this can be achieved for the CMTF-OPT algorithm. In any case, this masking approach is very important because besides enabling the analysis of incomplete datasets, it can also serve as stepping stone for estimating those missing values: When obtaining a low rank model of the incomplete dataset after masking the missing values, we can reconstruct the original data and our reconstruction will contain imputations for the missing values. This can be seen as the first iteration of an Expectation-Maximization scheme introduced in [TB05].

Following the formulation of [ADK11, AKD11] we define a 'weight' tensor $\underline{\mathbf{W}}$ which has '0' in all coefficients where values are missing, and '1' everywhere else. Similarly, we introduce three weight matrices $\mathbf{W}_i$ for each of the coupled matrices $\mathbf{Y}_i$. Then, the optimization function of the CMTF model becomes

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{D},\mathbf{E},\mathbf{G}} \|\underline{\mathbf{W}} * \left(\underline{\mathbf{X}} - \sum_k \mathbf{a}_k \circ \mathbf{b}_k \circ \mathbf{c}_k\right)\|_F^2 +$$
$$\|\mathbf{W}_1 * \left(\mathbf{Y}_1 - \mathbf{A}\mathbf{D}^T\right)\|_F^2 + \|\mathbf{W}_2 * \left(\mathbf{Y}_2 - \mathbf{B}\mathbf{E}^T\right)\|_F^2 + \|\mathbf{W}_3 * \left(\mathbf{Y}_3 - \mathbf{C}\mathbf{G}^T\right)\|_F^2$$

As we show in Algorithm 3.7, we may solve CMTF by solving six least squares problems in an alternating fashion. A fortuitous implication of this fact is that in order to handle missing values for CMTF, it suffices to solve

$$\min_{\mathbf{B}} \|\mathbf{W} * \left(\mathbf{X} - \mathbf{A}\mathbf{B}^T\right)\|_F^2 \tag{5.6}$$

where $\mathbf{W}$ is a weight matrix in the same sense as described a few lines earlier.

Our solution draws from a similar derivation in [PSB13] (Section 3.B) On our way tackling the above problem, we first need to investigate its scalar case, i.e. the case where we are interested only in $\mathbf{B}(j,f)$ for a fixed pair of j and f. The optimization problem may be rewritten as

$$\min_{\mathbf{B}(j,f)} \|\mathbf{W}(:,j) * \mathbf{X}(:,j) - \left(\mathbf{W}(:,j) * \mathbf{A}(:f)\right)\mathbf{B}(j,f)^T\| \tag{5.7}$$

which is essentially a scalar least squares problem of the form: $\min_b \|\mathbf{x} - \mathbf{a}b\|_2^2$ with solution in analytical form: $b = \frac{\mathbf{x}^T\mathbf{a}}{\|\mathbf{a}\|_2^2}$. The reader may notice that the dimensions of $\mathbf{W}$ and $\mathbf{A}$ are incompatible, however, in the scalar case of Eq. 5.7, we multiply element-wise the columns of $\mathbf{W}$ and $\mathbf{A}$, which are of the same dimensions.

We may, thus, solve this problem of Equation 5.6 using coordinate descent, on $\mathbf{B}$ iteratively, until convergence. Therefore, with the aforementioned derivation, we are able to modify our original algorithm in order to take missing values into account.

So far, we have described how to make ALS robust against missing values, however a few modifications to TURBO-SMT are required in order to be able to use the robust ALS core solver. In particular, in order to use the masking scheme for TURBO-SMT, during sampling, we have to take into account that some values in the data are missing: when we compute the weights for the biased sampling (i.e. Equations 5.2, 5.3, 5.4) we have to hide the missing values. In order to do that, we compute the weights on

$$\underline{\mathbf{X}}_m = \underline{\mathbf{W}} * \underline{\mathbf{X}}$$

and

$$\mathbf{Y}_{1m} = \mathbf{W}_1 * \mathbf{Y}_1$$

where the weights $\underline{\mathbf{W}}$ and $\mathbf{W}_1$ are defined above (and accordingly for $\mathbf{Y}_2, \mathbf{Y}_3$). This is equivalent to assuming that the missing values are equal to 0, and because computation of the weights is calculated through addition, treating the missing values as 0 in this context is equivalent to ignoring them. After these modifications, TURBO-SMT can be used along with the robust to missing values ALS algorithm for large and incomplete datasets.

5.5 Experimental Evaluation

5.5.1 Experimental Setup

We implemented TURBO-SMT in Matlab. Our implementation of the code is publicly available.[2] For the parallelization of the algorithm, we used Matlab's Parallel Computing Toolbox. For tensor manipulation, we used the Tensor Toolbox for Matlab [BK+15] which is optimized especially for sparse tensors (but works very well for dense ones too). We use the CMTF-ALS and the CMTF-OPT [AKD11] algorithms as baselines, i.e. we compare TURBO-SMT when using one of those algorithms as their core CMTF implementation, against the plain execution of those algorithms. In order to make this comparison possible, all datasets we used were within the capabilities of the Tensor Toolbox for Matlab.

We implemented our version of the ALS algorithm, and we used the CMTF Toobox[3] implementation of CMTF-OPT. All experiments were carried out on a machine with 4 Intel Xeon E74850 2.00GHz and 512Gb of RAM. The version of Matlab was R2013a (8.1.0.604) 64-bit. In the appendix, we show timing experiments on a less powerful machine, in order to demonstrate that the principle behind TURBO-SMT is applicable to a variety of platforms. Whenever we conducted multiple iterations of an experiment (due to the randomized nature of TURBO-SMT), we report error-bars along the plots. For all the following experiments we used either portions of the dataset analyzed in Chapter 9 (henceforth referred to as BRAINQ dataset), or the whole dataset.

5.5.2 Run time

As we have already discussed in the Introduction and shown in Fig. 10.1, TURBO-SMT achieved a speedup of 50-200 on the BRAINQ dataset; For all cases, the approximation

[2] http://www.cs.cmu.edu/~epapalex/src/turbo_smt.zip
[3] http://www.models.life.ku.dk/joda/CMTF_Toolbox

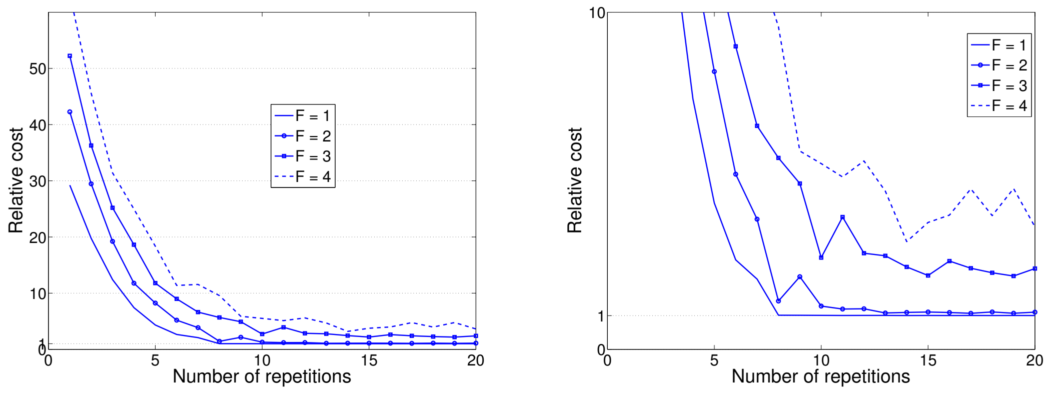
cost is either same as the baselines, or is larger by a small factor, indicating that TURBO-SMT is both fast and accurate. Key facts that contribute to this observed speedup are: 1) dimensionality reduction through sampling, 2) the fact that TURBO-SMT operates on sparse data throughout its lifetime, and 3) that TURBO-SMT is *highly parallelizable*. Figure 5.1 illustrates this behaviour. It is crucial to note that the speedup achieved is very significant: The ALS algorithm required more than 24 *hours* to be computed, and the CMTF-OPT algorithm took about 12 *hours*; TURBO-SMT was able to successfully *boost* both algorithms, while being almost as accurate. In the appendix, Section 5.A.1 we include more detailed experiments for the interested reader.

5.5.3 Accuracy

In this sub-section, we evaluate how accurately TURBO-SMT approximates the original data. In Figure 5.3 we demonstrate that the algorithm operates correctly, in the sense that it reduces the model error (Equation 5.1) when doing more repetitions. In particular, the vertical axis displays the relative error, i.e. $\frac{\text{TURBO-SMT error}}{\text{baseline error}}$ (with ideal being equal to 1) and the horizontal axis is the number of repetitions in the sampling. We use $60 \times 200 \times 9$ portion of the BRAINQ tensor and the entire matrix, to ensure that both the baseline (CMTF-OPT in this case) and TURBO-SMT run in a short amount of time. We set $s = \begin{bmatrix} 10 & 20 & 1 \end{bmatrix}$. We run the experiment 1000 times and we keep the solutions achieving the minimum error. As Figure 5.3 shows, TURBO-SMT's relative error decreases as a function of the repetitions, indicating that TURBO-SMT generally reduces the CMTF approximation error with every repetition. As the decomposition rank increases, the number of repetitions needed increases as well, since the model to be captured becomes more complex. Because we are operating on smaller pieces of data, we aim for very low rank decompositions, in order to avoid overfactoring (i.e. choosing a higher number than the rank of the data) which may cause problems with the stitching and lead to instabilities. Additionally, due to the fact that TURBO-SMT operates on random samples of the data, convergence of the approximation error is noisier than deterministic methods such as ALS.

It is important to note that given a large dataset, during the first few repetitions of TURBO-SMT, the accuracy may be comparable to the null model's (i.e. the all-zero model), as shown in Figure 5.3; this is because TURBO-SMT starts with a null model and progressively builds it up. However, the important take home point here is that given a machine or a cluster of machines with enough amount of cores, we can enable the baselines to work on very large datasets that may not fit in main memory, and by running more repetitions, we explore more of the data and improve the approximation accuracy of the model.

In the appendix of this chapter (Sec. 5.A.2) we have included additional results on the accuracy of TURBO-SMT for the interested reader.



(a) Full range of relative error (b) Magnified for clarity

Figure 5.3: **TURBO-SMT generally reduces the approximation error of the CMTF model**: The relative error of the model, as a function of the number of repetitions r is decreasing.

5.5.4 Sparsity

One of the main advantages of TURBO-SMT is that, it is triple-sparse, i.e. starting from (possibly) sparse data, every intermediate result of TURBO-SMT is sparse, as well as the final output. In Fig. 5.4 we demonstrate the sparsity of TURBO-SMT's results by introducing the relative sparsity metric; this intuitive metric is simply the ratio of the output size of the baseline algorithm, divided by the output size of TURBO-SMT. The output size is simply calculated by adding up the number of non-zero entries for all factor matrices output by the algorithm. We use a portion of the BRAINQ dataset in order to execute this experiment. We can see that for the relatively dense BRAINQ dataset, we obtained significantly more sparse results;, e.g. up to 65 times more sparse with almost same approximation error, for the case of CMTF-OPT. We observe a large difference of result sparsity when using CMTF-OPT, as opposed to ALS; most likely, this difference is due to the fact that, according to [AKD11], CMTF-OPT converges to a better local minimum than ALS. The results of Fig. 5.4 indicate that our triple-sparse algorithm is able to capture the most useful variation of the data, successfully suppressing noise.

5.5.5 Comparison to TUCKER Compression

One existing approach used for speeding up the PARAFAC decomposition is based on the CANDELINC theorem [CPK80] and has been used in [BA98, BSG99]; roughly, the method first uses TUCKER in order to compress the original tensor to a smaller, core tensor, then fits the PARAFAC decomposition on the core tensor, and finally, projects the factor matrices to the original space. In Section 5.3 of [KB09], it is implied that one could do the same for the CMTF problem. Before proceeding with a brief sketch of the approach which, to the best of our knowledge, *has not been published yet*, we provide a brief overview of the TUCKER decomposition.

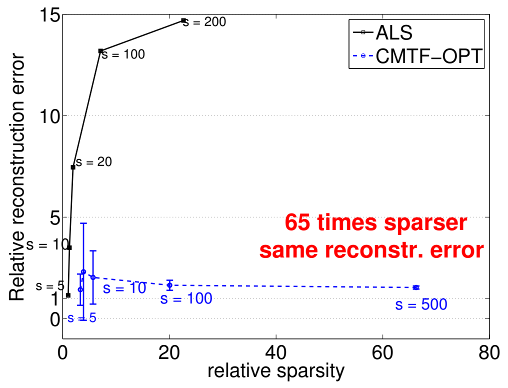


Figure 5.4: **Up to 65x sparser results for same accuracy**: The relative output size vs. the relative cost indicates that, even for very dense datasets such as BRAINQ, we are able to get up to 65 fold (for CMTF-OPT) decrease in the output size, while maintaining almost same approximation error as the baseline.

Consider the $I \times J \times K$ tensor $\underline{\mathbf{X}}$. Then, its $\{Q, R, P\}$ TUCKER decomposition consists of a $P \times Q \times R$ core tensor, say, $\underline{\mathbf{G}}$ and three assorted, unitary, matrices $\mathbf{U}, \mathbf{V}, \mathbf{Z}$ with sizes $I \times P$, $J \times Q$ and $K \times R$ respectively. The TUCKER objective function is:

$$\min_{\underline{\mathbf{G}},\mathbf{A},\mathbf{B},\mathbf{C}} \|\underline{\mathbf{X}} - \sum_{p=1}^{P}\sum_{q=1}^{Q}\sum_{r=1}^{R}\underline{\mathbf{G}}(p,q,r)\mathbf{u}_p \circ \mathbf{v}_q \circ \mathbf{z}_r\|_F^2$$

and we may write the decomposition, compactly, as:

$$\underline{\mathbf{X}} \approx \left[\underline{\mathbf{G}}^{(P\times Q\times R)}, \mathbf{U}^{(I\times P)}, \mathbf{V}^{(J\times Q)}, \mathbf{Z}^{(K\times R)}\right]$$

Having a tensor $\underline{\mathbf{X}}$ coupled with matrices $\mathbf{Y}_i,\ i = 1\cdots 3$, we may first obtain the TUCKER decomposition of $\mathbf{X}$. Consequently, we may use $\mathbf{U}$ in order to project $\mathbf{Y}_1$ to the compressed space, and respectively $\mathbf{V}$ for $\mathbf{Y}_2$ and $\mathbf{Z}$ for $\mathbf{Y}_3$. We, thus, obtain a new set of coupled data: the core tensor $\underline{\mathbf{G}}$, and the projected side matrices. Then, we fit a CMTF model to the compressed data, and as a final step, we use $\mathbf{U}$ in order to project $\mathbf{A}, \mathbf{D}$ to their original dimension (and accordingly for the rest of the factor matrices).

This method, however, lacks a few key features that TURBO-SMT has:

- TUCKER is now a bottleneck; its computation (even though there exist memory efficient implementations in the literature [BK+15], [KS08], which we use in our implementation) is very costly, compared to the simple sampling scheme that TURBO-SMT is using. Even though alternatives to TUCKER could be used for the compression (e.g. HOSVD [DLDMV00]), the authors of [BA98] state that the algorithm is better off using TUCKER compression, quality-wise, whenever TUCKER is able to be computed.

- This method, in contrast to TURBO-SMT, is not parallelizable, at least not in an obvious way, that would make its computation more efficient.
- This compression-based technique is not triple-sparse: The output of TUCKER is dense, hence the core tensor $\underline{\mathbf{G}}$ and the projected side matrices are going to be dense. Additionally, both ALS and CTMF-OPT [AKD11] produce dense factors. Therefore, this technique is prone to storage and interpretability issues.

We implemented this compression-based technique, using Tensor Toolbox's [BK+15] memory efficient implementation of the TUCKER decomposition. In Fig. 5.5, we illustrate the wall-clock time of this approach, compared to TURBO-SMT, on the entire BRAINQ dataset; we chose $s = 5$ for TURBO-SMT, and we chose $P = Q = R = 60$ for the compression-based technique. We observe that TURBO-SMT performs significantly better, while, additionally, producing sparse outputs.

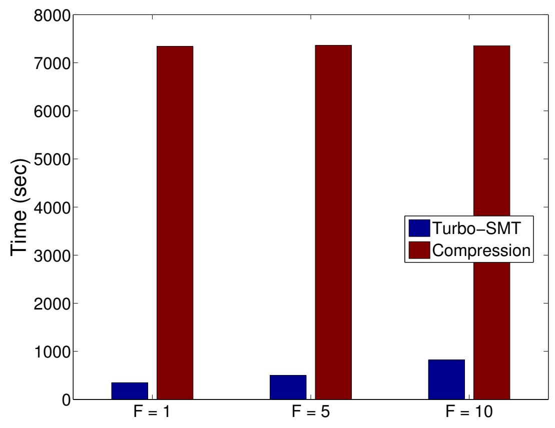


Figure 5.5: **TURBO-SMT significantly outperforms TUCKER-based compression method**: Comparison of TURBO-SMT and a compression-based technique, that uses the TUCKER decomposition, as described in Sec. 5.5.5.

5.5.6 Performance of ALS with missing values

In order to measure resilience to missing values we define the *Signal-to-Noise Ratio* (SNR) as simply as SNR $= \frac{\|\underline{\mathbf{X}}_m\|_F^2}{\|\underline{\mathbf{X}}_m - \underline{\mathbf{X}}_0\|_F^2}$, where $\underline{\mathbf{X}}_m$ is the reconstructed tensor when a m fraction of the values are missing. In Figure 5.6, we demonstrate the results of that experiment; we observe that even for a fair amount of missing data, the algorithm performs reasonably well, achieving high SNR. Moreover, for small amounts of missing data, the speed of the algorithm is not degraded, while for larger values, it is considerably slower, probably due to Matlab's implementation issues. However, this is encouraging, in the sense that if the amount of missing data is not overwhelming, TURBO-SMT is able to deliver a very good approximation of the latent subspace. This experiment was, again, conducted on a portion of BRAINQ.

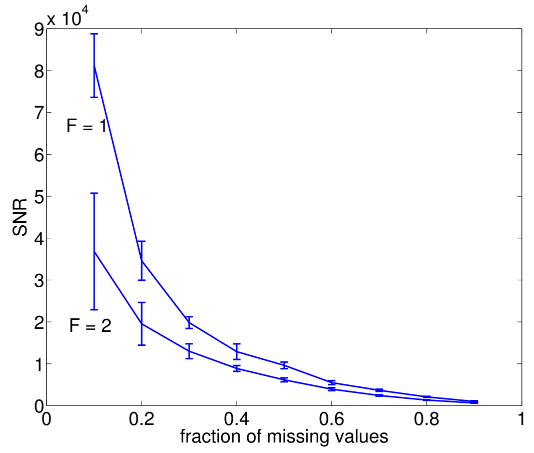


Figure 5.6: **TURBO-SMT handles missing values**: This Figure shows the Signal-to-Noise ratio (SNR)-as defined in the main text- as a function of the percentage of missing values. We can observe that, even for a fair amount of missing values, the SNR is quite high, signifying that TURBO-SMT is able to handle such ill-conditioned settings, with reasonable fidelity.

5.6 Conclusions

The main contributions of our work are:

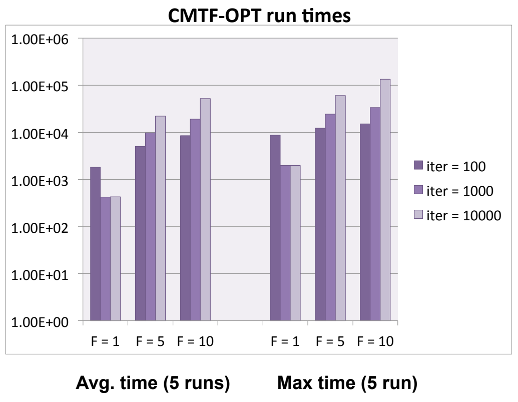
- *Parallel & triple-sparse algorithm:* TURBO-SMT is able to parallelize *any* CMTF solver, producing much sparser results. Moreover, the way that TURBO-SMT is defined, it can operate on data that do not fit in main memory, thus enabling state of the art CMTF solvers to work on such data, even though they are not specifically designed to do so.
- *Reproducibility*: We make our code public, enabling reproducibility and re-usability of our work.

Appendix: Additional Experimental Results

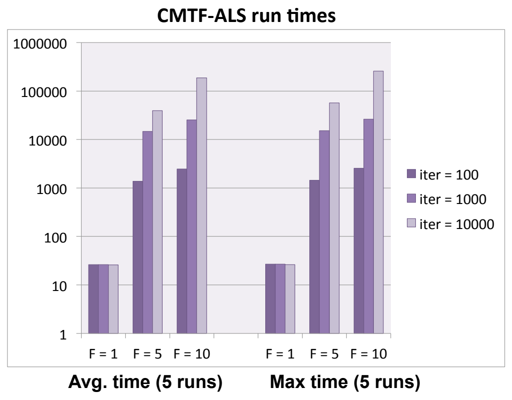
5.A.1 Run-time

In this section, we measure the execution time on the BRAINQ data for a variety of different configurations of TURBO-SMT, and for two different baselines (CMTF-OPT and CMTF-ALS). In particular, while keeping the number of repetitions r fixed to 4, we vary the sampling factor s for values $\{2, 5, 10, 20\}$. Because BRAINQ is highly imbalanced, the actual sampling factors we use are $\left[\frac{s}{2} \quad s \quad 1\right]$. We also vary the decomposition rank F for 1, 5, and 10.

The reason why we keep r fixed to 4 is because we conducted the same experiment on a machine with 4 cores (showing the results in the appendix) and for consistency purposes we chose the same r for both experiments. In Section 5.5.3 we show, however, that increasing r is improving the accuracy and if we use a machine with as many cores

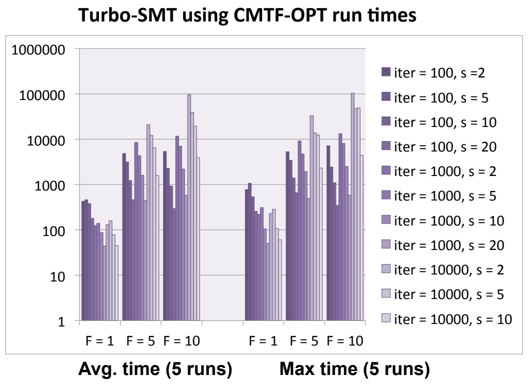


(a) CMTF_OPT

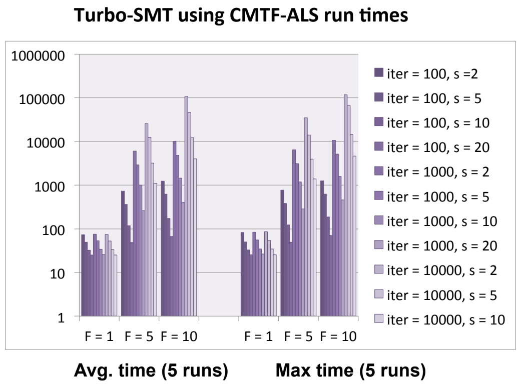


(b) CMTF_ALS

Figure 5.7: Run time (in seconds) for the baselines, on the full BRAINQ dataset. The results shown are over 5 full runs of the algorithm.



(a) TURBO-SMT using CMTF_OPT



(b) TURBO-SMT using CMTF_ALS

Figure 5.8: Run time (in seconds) for TURBO-SMT when using each of the baselines as core solver. The results shown are over 5 full runs of the algorithm. For every value of F, the bars on the plot starting from left to right, correspond to the parameters shown on the legend. The number of repetitions r was set to 4.

as r, then we may do so entirely in parallel, without significantly deviating from the timings that we report in this section.

When we compare TURBO-SMT against one of the two baselines, we make sure that we use the corresponding baseline as the core solver for TURBO-SMT. We observed a certain variation of solutions achieved by the baselines, depending on the number of iterations of the algorithm. Thus, we chose to experiment for three different limits for the number of iterations for the baselines (also enforcing the same limit in the core solver of TURBO-SMT). In particular, all the experiments are executed for 100, 1000, and

10000 iterations [4]. Note that this number indicates the *maximum* number of iterations (denoted by `iter`), however the baseline algorithm may terminate before reaching this number of iterations, depending on its (random) initialization. The termination criterion for both baselines was either meeting the maximum number of iterations `iter`, or the difference of the objective function between two consecutive iterations to be less than 10^{-8}. Finally, for every combination of rank, sampling factor, and number of iterations, we run TURBO-SMT and the corresponding baseline 5 times, in order to account for multiple local minima as well as variation in the execution times.

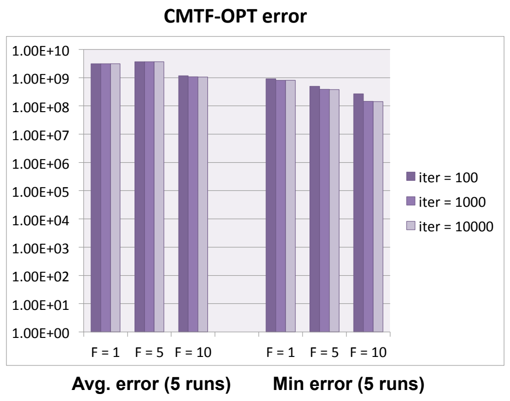
Figure 5.7 shows the run times for CMTF-OPT and CMTF-ALS. In particular, we show the average run time, as well as the maximum run time, in order to account for the worst case scenario. As expected, the run time for both baselines increases as the decomposition rank, and the number of iterations increase. There are a few cases where the runtime for `iter`=1000 is larger than that of `iter`=10000, however, since `iter` is merely the *maximum* number of iterations, and the problem being solved is a highly non-convex one, it is very likely that a run from the batch where `iter` was 10000 converged faster than a run from the batch of `iter`=1000. In Figure 5.8 we show the corresponding run times for TURBO-SMT, using both baselines. We observe the general trend of the run time increasing with the rank and the number of iterations, however, the run time decreases as the sampling factor decreases.

5.A.2 Accuracy

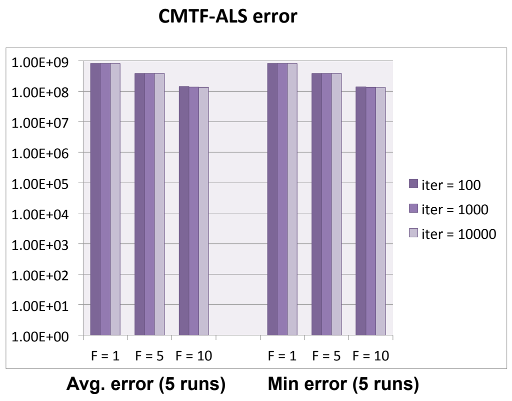
During the experiment described in Section 5.A.1, in addition to run time, we also measured the reconstruction error for each method and each combination of parameters. In Figure 5.9 we show the reconstruction errors for the two baselines, and in Figure 5.10 the error for TURBO-SMT. A first observation is that the error is fairly high for both the baselines and TURBO-SMT, fact that possibly indicates that the structure of the data is not exactly the one being imposed by CMTF, however, a low rank approximation is still useful for exploratory analysis.

Figures 5.10 assumes that the number of repetitions r is fixed to 4, which is a rather small number, considering the size of the data. When we, however, increase r, the approximation improves.

[4] 10000 iterations is the default for CMTF-OPT, according to the documentation.

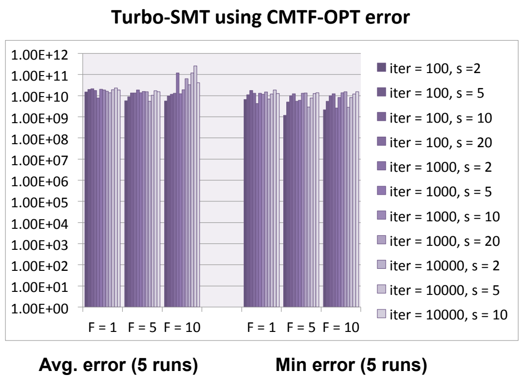


(a) CMTF_OPT

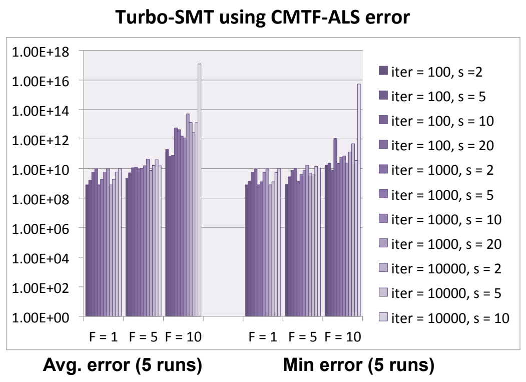


(b) CMTF_ALS

Figure 5.9: Reconstruction error of the tensor and the matrix (as measured by the objective function of the problem) for the two baselines. for $r = 4$.



(a) TURBO-SMT using CMTF_OPT



(b) TURBO-SMT using CMTF_ALS

Figure 5.10: Reconstruction error for TURBO-SMT.

Chapter based on material published in [SPF14].

Chapter 6

PARACOMP: A Parallel Algorithm for Big Tensor Decomposition Using Randomly Compressed Cubes

Parallelizing PARAFAC with identifiability guarantees using random compression.

In Chapters 4 and 5 we presented PARCUBE and TURBO-SMT which work remarkably well in practice and we provide guarantees for merging correctness. In this Chapter, we introduce PARACOMP, a novel architecture for parallel and distributed computation of low-rank tensor decomposition that is based on parallel processing of a set of randomly compressed replicas of the big tensor, which additionally provides *identifiability guarantees* of the final result.

6.1 Introduction

Tensors are becoming increasingly important, especially for analyzing big data, and tensors easily turn really big, e.g., $1000 \times 1000 \times 1000 = 1$ billion entries. Memory issues related to tensor computations with large but sparse tensors have been considered in [BK07], [KS08], and incorporated in the Tensor Toolbox for Matlab [BK^{+}15] which specializes in handling sparse tensors. The main idea in those references is to avoid intermediate product “explosion” when computing sequential tensor - matrix (“mode”) products, but the assumption is that the entire tensor fits in memory, and the mode products expand (as opposed to reduce) the size of the 'core' array that they are multiplied with. Adaptive tensor decomposition algorithms for cases where the data is serially acquired (or 'elongated') along one mode have been developed in [NS09], but these assume that the other two modes are relatively modest in size. More recently, a divide-and-conquer approach for decomposing big tensors has been proposed in [PC11]. The idea of [PC11] is to break the data in smaller 'boxes' which can be factored independently, and the results subsequently concatenated using an iterative process. This assumes that each smaller box admits a unique factorization (which cannot be guaranteed from 'global' uniqueness conditions alone), requires reconciling the different column permutations and scalings of the different blocks, and entails significant communication and signaling overhead.

All of the aforementioned techniques require that the full data be stored in (possibly distributed) memory. Realizing that this is a show-stopper for truly big tensors, in Chapter 4 we proposed PARCUBE, a random sampling approach, wherein judiciously sampled *significant* parts of the tensor are independently analyzed, and a common piece of data is used to anchor the different permutations and scalings. PARCUBE works very well in practice for sparse tensors and provides theoretical guarantees for merging correctness as we saw in Chapter 4, however, does not offer identifiability guarantees.

A different approach was taken in [SK12], which proposed *randomly* compressing a big tensor down to a far smaller one. Assuming that the big tensor admits a low-rank decomposition with sparse latent factors, such a random compression guarantees identifiability of the low-rank decomposition of the big tensor from the low-rank decomposition of the small tensor. This result can be viewed as a generalization of compressed sensing ideas from the linear to the multi-linear case. Still, this approach works only when the latent low-rank factors of the big tensor are known to be sparse - and this is often not the case.

This Chapter considers appropriate compression strategies for big (sparse or dense) tensors that admit a low-rank decomposition, whose latent factors need not be sparse. Latent sparsity is usually associated with membership problems such as clustering and co-clustering [PSB13]. A novel architecture for parallel and distributed computation of low-rank tensor decomposition that is especially well-suited for big tensors is proposed. The new architecture is based on parallel processing of a set of randomly compressed, reduced-size 'replicas' or the big tensor. Each replica is independently decomposed, and the results are joined via a master linear equation per tensor mode. The approach enables massive parallelism with guaranteed identifiability properties: if the big tensor is indeed of low rank and the system parameters are appropriately chosen, then the rank-one factors of the big tensor will indeed be recovered from the analysis of the reduced-size replicas. Furthermore, the architecture affords memory / storage and complexity gains of order $\frac{IJ}{F}$ for a big tensor of size $I \times J \times K$ of rank F with $F \leq I \leq J \leq K$. No sparsity is required in the tensor or the underlying latent factors, although such sparsity can be exploited to improve memory, storage and computational savings.

Notation: The notation used here is the same as the one shown in Table 2.1 of Chapter 2, but we repeat and augment it here for the purposes of this chapter, and for readability purposes.

A scalar is denoted by an italic letter, e.g. a. A column vector is denoted by a bold lowercase letter, e.g. $\mathbf{a}$ whose i-th entry is $\mathbf{a}(i)$. A matrix is denoted by a bold uppercase letter, e.g., $\mathbf{A}$ with (i,j)-th entry $\mathbf{A}(i,j)$; $\mathbf{A}(:,j)$ ($\mathbf{A}(i,:)$) denotes the j-th column (resp. i-th row) of $\mathbf{A}$. A tensor (three-way array) is denoted by an underlined bold uppercase letter, e.g., $\underline{\mathbf{X}}$, with (i,j,k)-th entry $\underline{\mathbf{X}}(i,j,k)$. $\underline{\mathbf{X}}(:,:,k)$ denotes the k-th frontal $I \times J$ matrix 'slab' of $\underline{\mathbf{X}}$, and similarly for the slabs along the other two modes. Vector, matrix and three-way array size parameters (mode lengths) are denoted by uppercase letters, e.g. I. $\circ$ stands for the vector outer product; i.e., for two vectors $\mathbf{a}$ $(I \times 1)$ and $\mathbf{b}$ $(J \times 1)$, $\mathbf{a} \circ \mathbf{b}$ is an $I \times J$ matrix with (i,j)-th element $\mathbf{a}(i)\mathbf{b}(j)$; i.e., $\mathbf{a} \circ \mathbf{b} = \mathbf{a}\mathbf{b}^T$. For three vectors, $\mathbf{a}$ $(I \times 1)$, $\mathbf{b}$ $(J \times 1)$, $\mathbf{c}$ $(K \times 1)$, $\mathbf{a} \circ \mathbf{b} \circ \mathbf{c}$ is an $I \times J \times K$ three-way array with (i,j,k)-th element $\mathbf{a}(i)\mathbf{b}(j)\mathbf{c}(k)$. The $\text{vec}(\cdot)$ operator stacks the columns of its matrix argument in one tall column; $\otimes$ stands for the Kronecker product; $\odot$ stands for the Khatri-Rao (column-wise Kronecker) product: given $\mathbf{A}$ $(I \times F)$ and $\mathbf{B}$ $(J \times F)$, $\mathbf{A} \odot \mathbf{B}$ is the $JI \times F$ matrix

$$\mathbf{A} \odot \mathbf{B} = \left[\mathbf{A}(:,1) \otimes \mathbf{B}(:,1) \cdots \mathbf{A}(:,F) \otimes \mathbf{B}(:,F)\right]$$

For a square matrix $\mathbf{S}$, $\text{Tr}(\mathbf{S})$ denotes its trace, i.e., the sum of elements on its main diagonal. $||\mathbf{x}||_2^2$ is the Euclidean norm squared, and $||\mathbf{A}||_F^2$, $||\underline{\mathbf{X}}||_F^2$ the Frobenius norm squared - the sum of squares of all elements of the given vector, matrix, or tensor.

6.2 Tensor Decomposition Preliminaries

Here, we revisit some of the relevant notions and definitions that we also outlined in Chapter 2.

Rank decomposition: The rank of an $I \times J$ matrix $\mathbf{X}$ is the smallest number of rank-one matrices (vector outer products of the form $\mathbf{a} \circ \mathbf{b}$) needed to synthesize $\mathbf{X}$ as

$$\mathbf{X} = \sum_{f=1}^{F} \mathbf{a}_f \circ \mathbf{b}_f = \mathbf{A}\mathbf{B}^T,$$

where $\mathbf{A} := [\mathbf{a}_1, \cdots, \mathbf{a}_F]$, and $\mathbf{B} := [\mathbf{b}_1, \cdots, \mathbf{b}_F]$. This relation can be expressed element-wise as

$$\mathbf{X}(i,j) = \sum_{f=1}^{F} \mathbf{a}_f(i)\mathbf{b}_f(j).$$

The rank of an $I \times J \times K$ three-way array $\underline{\mathbf{X}}$ is the smallest number of outer products needed to synthesize $\underline{\mathbf{X}}$ as

$$\underline{\mathbf{X}} = \sum_{f=1}^{F} \mathbf{a}_f \circ \mathbf{b}_f \circ \mathbf{c}_f.$$

This relation can be expressed element-wise as

$$\underline{\mathbf{X}}(i,j,k) = \sum_{f=1}^{F} \mathbf{a}_f(i)\mathbf{b}_f(j)\mathbf{c}_f(k).$$

In the sequel we will assume that F is minimal, i.e., $F = \text{rank}(\underline{\mathbf{X}})$, unless otherwise noted. The tensor $\underline{\mathbf{X}}$ comprises K 'frontal' slabs of size $I \times J$; denote them $\{\mathbf{X}_k\}_{k=1}^{K}$, with $\mathbf{X}_k := \underline{\mathbf{X}}(:,:,k)$. Re-arranging the elements of $\underline{\mathbf{X}}$ in a tall matrix $\mathbf{X} := [\text{vec}(\mathbf{X}_1), \cdots, \text{vec}(\mathbf{X}_K)]$, it can be shown that

$$\mathbf{X} = (\mathbf{B} \odot \mathbf{A})\mathbf{C}^T \Longleftrightarrow \mathbf{x} := \text{vec}(\mathbf{X}) = (\mathbf{C} \odot \mathbf{B} \odot \mathbf{A})\,\mathbf{1},$$

where, $\mathbf{A}$, $\mathbf{B}$ are as defined for the matrix case, $\mathbf{C} := [\mathbf{c}_1, \cdots, \mathbf{c}_F]$, $\mathbf{1}$ is a vector of all 1's, and we have used the vectorization property of the Khatri-Rao product $\text{vec}(\mathbf{A}\mathbf{D}(\mathbf{d})\mathbf{B}^T) = (\mathbf{B} \odot \mathbf{A})\,\mathbf{d}$, where $\mathbf{D}(\mathbf{d})$ is a diagonal matrix with the vector $\mathbf{d}$ as its diagonal.

PARAFAC: The above rank decomposition model for tensors is known as *parallel factor analysis* (PARAFAC) [Har70, Har72a] or *canonical decomposition* (CANDECOMP) [CC70b], or CP (and CPD) for CANDECOMP-PARAFAC (Decomposition), or *canonical polyadic decomposition* (CPD, again), as we introduced in Section 2.3.1. PARAFAC is usually fitted using an alternating least squares procedure based on the model equation $\mathbf{X} = (\mathbf{B} \odot \mathbf{A})\mathbf{C}^T$. In practice we will have $\mathbf{X} \approx (\mathbf{B} \odot \mathbf{A})\mathbf{C}^T$, due to measurement noise and other imperfections, or simply because we wish to approximate a higher-rank model with a lower-rank one. Fixing $\mathbf{A}$ and $\mathbf{B}$, we solve

$$\min_{\mathbf{C}} ||\mathbf{X} - (\mathbf{B} \odot \mathbf{A})\mathbf{C}^T||_F^2,$$

which is a linear least squares problem. We can bring any of the matrix factors to the right by reshuffling the data, yielding corresponding conditional updates for $\mathbf{A}$ and $\mathbf{B}$. We can revisit each matrix in a circular fashion until convergence of the cost function, and this is the most commonly adopted aproach to fitting the PARAFAC model, in good part because of its conceptual and programming simplicity, plus the ease with which we can incorporate additional constraints on the columns of $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ [BS98].

TUCKER: PARAFAC is in a way the most basic tensor model, because of its direct relationship to tensor rank and the concept of rank decomposition; but other algebraic tensor models exist, and the most notable one is known as TUCKER, as also seen in Section 2.3.2. Like PARAFAC, TUCKER is a sum of outer products model, involving outer products of columns of three matrices, $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$. Unlike PARAFAC however, which restricts interactions to corresponding columns (so that the first column of $\mathbf{A}$ only appears in one outer product involving the first column of $\mathbf{B}$ and the first column of $\mathbf{C}$), TUCKER includes all outer products of every column of $\mathbf{A}$ with every column of $\mathbf{B}$ and every column of $\mathbf{C}$. Each such outer product is further weighted by the corresponding entry of a so-called *core tensor*, whose dimensions are equal to the number of columns of $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$.

Consider again the $I \times J \times K$ three-way array $\underline{\mathbf{X}}$ comprising K matrix slabs $\{\mathbf{X}_k\}_{k=1}^K$, arranged into the tall matrix $\mathbf{X} := [\text{vec}(\mathbf{X}_1), \cdots, \text{vec}(\mathbf{X}_K)]$. The Tucker3 model can be written in matrix form as

$$\mathbf{X} \approx (\mathbf{B} \otimes \mathbf{A})\mathbf{G}\mathbf{C}^T,$$

where $\mathbf{G}$ is the core tensor in matrix form, and $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ can be assumed orthogonal without loss of generality, because linear transformations of $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ can be absorbed in $\mathbf{G}$. The non-zero elements of the core tensor determine the interactions between columns of $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$. The associated model-fitting problem is

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{G}} ||\mathbf{X} - (\mathbf{B} \otimes \mathbf{A})\mathbf{G}\mathbf{C}^T||_F^2,$$

which is usually solved using an alternating least squares procedure. The TUCKER model can be fully vectorized as $\text{vec}(\mathbf{X}) \approx (\mathbf{C} \otimes \mathbf{B} \otimes \mathbf{A})\,\text{vec}(\mathbf{G})$.

Identifiability: The distinguishing feature of the PARAFAC model is its essential uniqueness: under certain conditions, $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ can be identified from $\mathbf{X}$ up to a common permutation and scaling / counter-scaling of columns [CC70b, Har70, Har72a, Kru77, SB00, JS04, SS07, CO12]. In contrast, Tucker3 is highly non-unique; the inclusion of all possible outer products of columns of $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ results in over-parametrization that renders it unidentifiable in most cases of practical interest. Still, TUCKER is useful as an exploratory tool and for data compression / interpolation; we will return to this shortly.

Consider an $I \times J \times K$ tensor $\underline{\mathbf{X}}$ of rank F. In vectorized form, it can be written as the $IJK \times 1$ vector $\mathbf{x} = (\mathbf{A} \odot \mathbf{B} \odot \mathbf{C})\,\mathbf{1}$, for some $\mathbf{A}$ $(I \times F)$, $\mathbf{B}$ $(J \times F)$, and $\mathbf{C}$ $(K \times F)$ - a PARAFAC model of size $I \times J \times K$ and order F parameterized by $(\mathbf{A}, \mathbf{B}, \mathbf{C})$. (Notice the slight abuse of notation: we switched from $\mathbf{x} = (\mathbf{C} \odot \mathbf{B} \odot \mathbf{A})\,\mathbf{1}$ to $\mathbf{x} = (\mathbf{A} \odot \mathbf{B} \odot \mathbf{C})\,\mathbf{1}$. The two are related via a row permutation, or by switching the roles of $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$.) The *Kruskal-rank* of $\mathbf{A}$, denoted $k_{\mathbf{A}}$, is the maximum k such that *any* k columns of $\mathbf{A}$ are linearly independent ($k_{\mathbf{A}} \le r_{\mathbf{A}} := \text{rank}(\mathbf{A})$).

Theorem 6.1.:
[Kru77] Given $\underline{\mathbf{X}}$ ($\Leftrightarrow$ $\mathbf{x}$), $(\mathbf{A}, \mathbf{B}, \mathbf{C})$ are unique up to a common column permutation and scaling (e.g., scaling the first column of $\mathbf{A}$ and counter-scaling the first column of $\mathbf{B}$ and/or $\mathbf{C}$, so long as their product remains the same), provided that $k_{\mathbf{A}} + k_{\mathbf{B}} + k_{\mathbf{C}} \ge 2F + 2$. An equivalent and perhaps more intuitive way to express this is that the outer products $\mathbf{a}_f \circ \mathbf{b}_f \circ \mathbf{c}_f$ (i.e., the rank-one factors of $\underline{\mathbf{X}}$) are unique.

Note that we can always reshuffle the order of these rank-one factors (e.g., swap $\mathbf{a}_1 \circ \mathbf{b}_1 \circ \mathbf{c}_1$ and $\mathbf{a}_2 \circ \mathbf{b}_2 \circ \mathbf{c}_2$) without changing their sum $\underline{\mathbf{X}} = \sum_{f=1}^F \mathbf{a}_f \circ \mathbf{b}_f \circ \mathbf{c}_f$, but this is a trivial and inherently unresolvable ambiguity that we will ignore in the sequel. Theorem 6.1 is Kruskal's celebrated uniqueness result [Kru77], see also follow-up work in [SB00, JS04, SS07]. Kruskal's result applies to given $(\mathbf{A}, \mathbf{B}, \mathbf{C})$, i.e., it can establish uniqueness of a given decomposition. Recently, more relaxed uniqueness conditions have been obtained, which only depend on the size and rank of the tensor - albeit they cover *almost* all tensors of the given size and rank, i.e., except for a set of measure zero. Two such conditions are summarized next.

Theorem 6.2.:
[StBDL06] (see also [JS04]) Consider an $I \times J \times K$ tensor $\underline{\mathbf{X}}$ of rank F. If

$$r_{\mathbf{C}} = F \text{ (which implies } K \geq F)$$

and

$$I(I-1)J(J-1) \geq 2F(F-1),$$

then the rank-one factors of $\underline{\mathbf{X}}$ are almost surely unique.

Theorem 6.3.:
[CO12] Consider an $I \times J \times K$ tensor $\underline{\mathbf{X}}$ of rank F. Order the dimensions so that $I \leq J \leq K$. Let i be maximal such that $2^i \leq I$, and likewise j maximal such that $2^j \leq J$. If $F \leq 2^{i+j-2}$, then the rank-one factors of $\underline{\mathbf{X}}$ are almost surely unique. For I, J powers of 2, the condition simplifies to $F \leq \frac{IJ}{4}$. More generally, the condition implies that if $F \leq \frac{(I+1)(J+1)}{16}$, then $\underline{\mathbf{X}}$ has a unique decomposition almost surely.

Before we proceed to discuss big data and cloud computing aspects of tensor decomposition, we state two lemmas from [SK12] which we will need in the sequel.

Lemma 6.1.:
[SK12] Consider $\tilde{\mathbf{A}} := \mathbf{U}^T\mathbf{A}$, where $\mathbf{A}$ is $I \times F$, and let the $I \times L$ matrix $\mathbf{U}$ be randomly drawn from an absolutely continuous distribution (e.g., multivariate Gaussian with a non-singular covariance matrix). Then $k_{\tilde{\mathbf{A}}} = \min(L, k_{\mathbf{A}})$ almost surely (with probability 1).

Lemma 6.2.:
[SK12] Consider $\tilde{\mathbf{A}} = \mathbf{U}^T\mathbf{A}$, where $\mathbf{A}$ $(I \times F)$ is deterministic, tall/square $(I \geq F)$ and full column rank $r_{\mathbf{A}} = F$, and the elements of $\mathbf{U}$ $(I \times L)$ are i.i.d. Gaussian zero mean, unit variance random variables. Then the distribution of $\tilde{\mathbf{A}}$ is absolutely continuous (nonsingular multivariate Gaussian).

6.3 Tensor Compression

When dealing with big tensors $\underline{\mathbf{X}}$ that do not fit in main memory, a reasonable idea is to try to compress $\underline{\mathbf{X}}$ to a much smaller tensor that somehow captures most of the systematic variation in $\underline{\mathbf{X}}$. The commonly used compression method is to fit a low-dimensional orthogonal Tucker3 model (with low mode-ranks) [SBGW04, Kro08], then regress the data onto the fitted mode-bases. This idea has been exploited in existing PARAFAC model-fitting software, such as COMFAC [BSG99], as a useful quick-and-dirty way to initialize alternating least squares computations in the uncompressed domain, thus accelerating convergence. A key issue with TUCKER compression of big tensors is that it requires computing singular value decompositions of the various matrix unfoldings of the full data, in an alternating fashion. This is a serious bottleneck for big data. Another issue is that Tucker3 compression is lossy, and it cannot guarantee that identifiability properties will be preserved. Finally, fitting a PARAFAC model to the compressed data

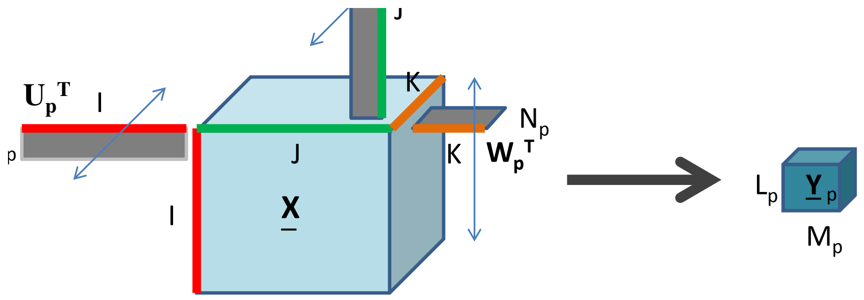


Figure 6.1: **Schematic illustration of tensor compression**: going from an $I \times J \times K$ tensor $\underline{\mathbf{X}}$ to a much smaller $L_p \times M_p \times N_p$ tensor $\underline{\mathbf{Y}}_p$ via multiplying (every slab of) $\underline{\mathbf{X}}$ from the I-mode with $\mathbf{U}_p^T$, from the J-mode with $\mathbf{V}_p^T$, and from the K-mode with $\mathbf{W}_p^T$, where $\mathbf{U}_p$ is $I \times L_p$, $\mathbf{V}_p$ is $J \times M_p$, and $\mathbf{W}_p$ is $K \times N_p$.

can only yield an approximate model for the original uncompressed data, and eventually decompression and iterations with the full data are required to obtain fine estimates.

Consider compressing $\mathbf{x}$ into $\mathbf{y} = \mathbf{S}\mathbf{x}$, where $\mathbf{S}$ is $d \times IJK$, $d \ll IJK$. Sidiropoulos & Kyrillidis [SK12] proposed using a specially structured compression matrix $\mathbf{S} = \mathbf{U}^T \otimes \mathbf{V}^T \otimes \mathbf{W}^T$, which corresponds to multiplying (every slab of) $\underline{\mathbf{X}}$ from the I-mode with $\mathbf{U}^T$, from the J-mode with $\mathbf{V}^T$, and from the K-mode with $\mathbf{W}^T$, where $\mathbf{U}$ is $I \times L$, $\mathbf{V}$ is $J \times M$, and $\mathbf{W}$ is $K \times N$, with $L \leq I$, $M \leq J$, $N \leq K$ and $LMN \ll IJK$; see Fig. 6.1. Such an $\mathbf{S}$ corresponds to compressing each mode individually, which is often natural, and the associated multiplications can be efficiently implemented as we show in Section 6.3.1 Due to a fortuitous property of the Kronecker product [Bre78],

$$\left(\mathbf{U}^T \otimes \mathbf{V}^T \otimes \mathbf{W}^T\right)(\mathbf{A} \odot \mathbf{B} \odot \mathbf{C}) = \left((\mathbf{U}^T\mathbf{A}) \odot (\mathbf{V}^T\mathbf{B}) \odot (\mathbf{W}^T\mathbf{C})\right),$$

from which it follows that

$$\mathbf{y} = \left((\mathbf{U}^T\mathbf{A}) \odot (\mathbf{V}^T\mathbf{B}) \odot (\mathbf{W}^T\mathbf{C})\right)\mathbf{1} = \left(\tilde{\mathbf{A}} \odot \tilde{\mathbf{B}} \odot \tilde{\mathbf{C}}\right)\mathbf{1}.$$

i.e., the compressed data follow a PARAFAC model of size $L \times M \times N$ and order F parametrized by $(\tilde{\mathbf{A}}, \tilde{\mathbf{B}}, \tilde{\mathbf{C}})$, with $\tilde{\mathbf{A}} := \mathbf{U}^T\mathbf{A}$, $\tilde{\mathbf{B}} := \mathbf{V}^T\mathbf{B}$, $\tilde{\mathbf{C}} := \mathbf{W}^T\mathbf{C}$.

6.3.1 Complexity of multi-way compression

Multiplying a dense $L \times I$ matrix $\mathbf{U}^T$ with a dense vector $\mathbf{a}$ to compute $\mathbf{U}^T\mathbf{a}$ has complexity LI. Taking the product of $\mathbf{U}^T$ and the first $I \times J$ frontal slab $\underline{\mathbf{X}}(:,:,1)$ of the $I \times J \times K$ tensor $\underline{\mathbf{X}}$ has complexity LIJ. Pre-multiplying from the left all frontal slabs of $\underline{\mathbf{X}}$ by $\mathbf{U}^T$ (computing a *mode product*) therefore requires $LIJK$ operations, when all operands are dense. Multi-way compression as in Fig. 6.1 comprises three mode products, suggesting a complexity of $LIJK + MLJK + NLMK$, if the first mode is compressed first, followed by the second, and then the third mode. Notice that the order in which the mode products are computed affects the complexity of the overall operation; but order-wise, this is

$O(\min(L, M, N)IJK)$. Also notice that if I, J, K are of the same order, and so are L, M, N, then the overall complexity is $O(LI^3)$.

If $\mathbf{a}$ is sparse with NZ($\underline{\mathbf{a}}$) nonzero elements, we can compute $\mathbf{U}^T\mathbf{a}$ as a weighted sum of the columns of $\mathbf{U}^T$ corresponding to the nonzero elements of $\mathbf{a}$. This reduces matrix-vector multiplication complexity to LNZ($\underline{\mathbf{a}}$). It easily follows that if $\underline{\mathbf{X}}$ has NZ ($\underline{\mathbf{X}}$) nonzero elements, the complexity of pre-multiplying from the left all frontal slabs of $\underline{\mathbf{X}}$ by $\mathbf{U}^T$ can be reduced to LNZ ($\underline{\mathbf{X}}$). The problem is that, after computing the first mode product, the resulting tensor will be dense! - hence subsequent mode products cannot exploit sparsity to reduce complexity. Note that, in addition to computational complexity, memory or secondary storage to save the intermediate results of the computation becomes an issue, even if the original tensor $\underline{\mathbf{X}}$ is sparse.

In scalar form, the (ℓ, m, n)-th element of the tensor $\underline{\mathbf{Y}}$ after multi-way compression can be written as

$$\underline{\mathbf{Y}}(\ell, m, n) = \sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K}\mathbf{U}(i,\ell)\mathbf{V}(j,m)\mathbf{W}(k,n)\underline{\mathbf{X}}(i,j,k)$$

Claim 6.1.:
Suppose that $\underline{\mathbf{X}}$ is sparse, with NZ ($\underline{\mathbf{X}}$) nonzero elements, and suppose that it is stored as a serial list with entries formatted as $[i, j, k, v]$, where v is the nonzero value at tensor position (i, j, k). Suppose that the list is indexed by an integer index s, i.e., $[i(s), j(s), k(s), v(s)]$ is the record corresponding to the s-th entry of the list. Then the following simple Algorithm 6.1 will compute the multi-way compressed tensor $\underline{\mathbf{Y}}$ in only LMNNZ ($\underline{\mathbf{X}}$) operations, requiring only LMN cells of memory to store the result, and $IL + JM + KN$ cells of memory to store the matrices **U**, **V**, **W**.

Algorithm 6.1: Efficient Multi-way Compression

```
Input: Tensor $\underline{\mathbf{X}}$ in list format with nonzero values in $v$(see text), compression matrices $\mathbf{U}, \mathbf{V}, \mathbf{W}$
    for each respective mode
Output: Randomly compressed tensor $\underline{\mathbf{Y}}$
1: for $\ell = 1 \cdots L$ do
2:     for $m = 1 \cdots M$ do
3:         for $n = 1 \cdots N$ do
4:             $\underline{\mathbf{Y}}(\ell, m, n) + \mathbf{U}(i(s), \ell)\mathbf{V}(j(s), m)\mathbf{W}(k(s), n)v(s)$
5:         end for
6:     end for
7: end for
```

Notice that, even if $\underline{\mathbf{X}}$ is dense (i.e., NZ ($\underline{\mathbf{X}}$) $= IJK$), the above algorithm only needs to read each element of $\underline{\mathbf{X}}$ once, so complexity will be $LMNIJK$ but memory will still be very modest: only LMN cells of memory to store the result, and $IL + JM + KN$ cells of memory to store the matrices **U**, **V**, **W**. Contrast this to the 'naive' way of serially computing the mode products, whose complexity order is $O(\min(L, M, N)IJK)$ but whose memory requirements are huge for dense $\mathbf{U}$, $\mathbf{V}$, $\mathbf{W}$, due to intermediate result explosion - even for sparse $\underline{\mathbf{X}}$. We see a clear complexity - memory trade-off between the two approaches for dense data, but Algorithm 1 is a clear winner for sparse data, because sparsity is lost after the first mode product. Notice that the above algorithm can be fully parallelized in several ways - by splitting the list of nonzero elements across cores or processors (paying in terms of auxiliary memory replications to store partial results for $\underline{\mathbf{Y}}$ and the matrices $\mathbf{U}$, $\mathbf{V}$, $\mathbf{W}$ locally at each processor), or by splitting the (ℓ, m, n) loops - at the cost of replicating the data list. As a final word, the memory access pattern (whether we read and write consecutive memory elements in blocks, or make wide 'strides') is the performance-limiting factor for truly big data, and the above algorithm makes strides in reading elements of $\mathbf{U}$, $\mathbf{V}$, $\mathbf{W}$, and writing elements of $\underline{\mathbf{Y}}$. There are ways to reduce these strides, at the cost of requiring more memory and more floating point operations.

6.4 Stepping-stone results

Given the result of Section 6.3, we are now interested in answering the following two questions:

1. Under what conditions on $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ and $\mathbf{U}$, $\mathbf{V}$, $\mathbf{W}$ are $(\tilde{\mathbf{A}}, \tilde{\mathbf{B}}, \tilde{\mathbf{C}})$ identifiable from $\mathbf{y}$?
2. Under what conditions, if any, are $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ identifiable from $(\tilde{\mathbf{A}}, \tilde{\mathbf{B}}, \tilde{\mathbf{C}})$?

We start by answering the first question in this section.

Theorem 6.4.:
Let $\mathbf{x} = (\mathbf{A} \odot \mathbf{B} \odot \mathbf{C})\,\mathbf{1} \in \mathbb{R}^{IJK}$, where $\mathbf{A}$ is $I \times F$, $\mathbf{B}$ is $J \times F$, $\mathbf{C}$ is $K \times F$, and consider compressing it to $\mathbf{y} = \left(\mathbf{U}^T \otimes \mathbf{V}^T \otimes \mathbf{W}^T\right)\mathbf{x} = \left((\mathbf{U}^T\mathbf{A}) \odot (\mathbf{V}^T\mathbf{B}) \odot (\mathbf{W}^T\mathbf{C})\right)\mathbf{1} = \left(\tilde{\mathbf{A}} \odot \tilde{\mathbf{B}} \odot \tilde{\mathbf{C}}\right)\mathbf{1} \in \mathbb{R}^{LMN}$, where the mode-compression matrices $\mathbf{U}$ ($I \times L, L \leq I$), $\mathbf{V}$ ($J \times M, M \leq J$), and $\mathbf{W}$ ($K \times N, N \leq K$) are independently drawn from an absolutely continuous distribution. If

$$\min(L, k_{\mathbf{A}}) + \min(M, k_{\mathbf{B}}) + \min(N, k_{\mathbf{C}}) \geq 2F + 2,$$

then $\tilde{\mathbf{A}}, \tilde{\mathbf{B}}, \tilde{\mathbf{C}}$ are almost surely identifiable from the compressed data $\mathbf{y}$ up to a common column permutation and scaling.

Proof. Theorem is a direct consequence of Lemma 6.1 and Kruskal's uniqueness condition in Theorem 6.1. ■

More relaxed conditions for identifiability of $\tilde{\mathbf{A}}, \tilde{\mathbf{B}}, \tilde{\mathbf{C}}$ can be derived from Lemma 6.2, and Theorems 6.2 and 6.3.

Theorem 6.5.:
For $\mathbf{x}$, $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{U}$, $\mathbf{V}$, $\mathbf{W}$, and $\mathbf{y}$ as in Theorem 6.4, if $F \leq \min(I, J, K)$, $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ are all full column rank (F), $N \geq F$, and

$$L(L-1)M(M-1) \geq 2F(F-1),$$

then $\tilde{\mathbf{A}}, \tilde{\mathbf{B}}, \tilde{\mathbf{C}}$ are almost surely identifiable from the compressed data $\mathbf{y}$ up to a common column permutation and scaling.

Proof. The proof follows by combining Lemma 6.2, and Theorems 6.2 and 6.3. ■

Theorem 6.6.:
For $\mathbf{x}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{U}, \mathbf{V}, \mathbf{W}$, and $\mathbf{y}$ as in Theorem 6.4, if $F \leq \min(I, J, K)$, **A**, **B**, **C** are all full column rank (F), $L \leq M \leq N$, and

$$(L+1)(M+1) \geq 16F,$$

then $\tilde{\mathbf{A}}, \tilde{\mathbf{B}}, \tilde{\mathbf{C}}$ are almost surely identifiable from the compressed data $\mathbf{y}$ up to a common column permutation and scaling.

Proof. The proof follows by the remark below:
Remark 6.1.:
$F \leq \min(I, J, K) \Rightarrow$ full column rank **A**, **B**, **C** almost surely, i.e., tall matrices are full column rank except for a set of measure zero. In other words, if $F \leq \min(I, J, K)$ and **A**, **B**, **C** are themselves considered to be independently drawn from an absolutely continuous distribution with respect to the Lebesgue measure in $\mathbb{R}^{IF}$, $\mathbb{R}^{JF}$, and $\mathbb{R}^{KF}$, respectively, then they will all be full column rank with probability 1.

■

6.5 Main results

Theorems 6.4, 6.5, 6.6 can establish uniqueness of $\tilde{\mathbf{A}}, \tilde{\mathbf{B}}, \tilde{\mathbf{C}}$, but we are ultimately interested in $\mathbf{A}, \mathbf{B}, \mathbf{C}$. We know that $\tilde{\mathbf{A}} = \mathbf{U}^T\mathbf{A}$, and we know $\mathbf{U}^T$, but, unfortunately, it is a fat matrix that cannot be inverted. In order to uniquely recover **A**, we need additional structural constraints. Sidiropoulos & Kyrillidis [SK12] proposed exploiting column-wise sparsity in **A** (and likewise $\mathbf{B}, \mathbf{C}$), which is often plausible in practice[1]. Sparsity is a powerful constraint, but it is not always valid (or a sparsifying basis may be unknown). For this reason, we propose here a different solution, based on creating and factoring a number of randomly reduced 'replicas' of the full data. The majority of the theoretical work in this section as it originally appears in [SPF14] is attributed to co-author Nicholas D. Sidiropoulos. Here we present a concise subset of those results that demonstrate the theoretical underpinnings behind PARACOMP.

Consider spawning P randomly compressed reduced-size "replicas" $\left\{\underline{\mathbf{Y}}_p\right\}_{p=1}^{P}$ of the tensor $\underline{\mathbf{X}}$, where $\underline{\mathbf{Y}}_p$ is created using mode compression matrices $(\mathbf{U}_p, \mathbf{V}_p, \mathbf{W}_p)$, see Fig. 6.2. Assume that identifiability conditions per Theorem 6.5 or Theorem 6.6 hold, so that $\tilde{\mathbf{A}}_p, \tilde{\mathbf{B}}_p, \tilde{\mathbf{C}}_p$ are almost surely identifiable (up to permutation and scaling of columns) from $\underline{\mathbf{Y}}_p$. Then, upon factoring $\underline{\mathbf{Y}}_p$ into F rank-one components, we obtain

$$\tilde{\mathbf{A}}_p = \mathbf{U}_p^T \mathbf{A} \mathbf{\Pi}_p \mathbf{\Lambda}_p, \tag{6.1}$$

[1]**A** need only be sparse with respect to (when expressed in) a suitable basis, provided the sparsifying basis is known *a priori*.

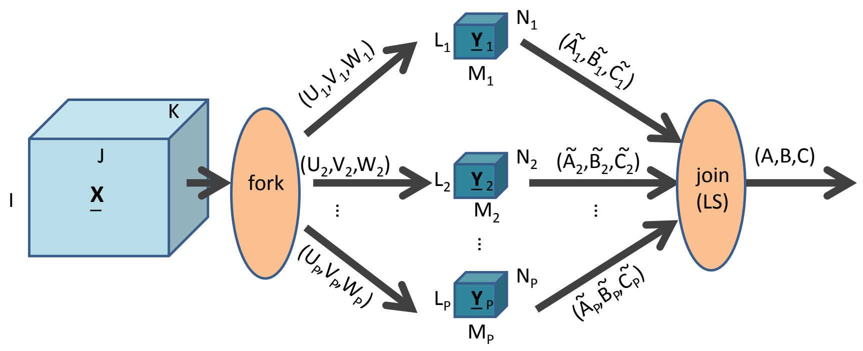


Figure 6.2: **Schematic illustration of the PARACOMP fork-join architecture**: The fork step creates a set of P randomly compressed reduced-size 'replicas' $\{\underline{\mathbf{Y}}_p\}_{p=1}^{P}$. Each $\underline{\mathbf{Y}}_p$ is obtained by applying $(\mathbf{U}_p, \mathbf{V}_p, \mathbf{W}_p)$ to $\underline{\mathbf{X}}$. Each $\underline{\mathbf{Y}}_p$ is then independently factored (all P threads can be executed in parallel). The join step collects the estimated mode loading sub-matrices $\left(\tilde{\mathbf{A}}_p, \tilde{\mathbf{B}}_p, \tilde{\mathbf{C}}_p\right)$ from the P threads, and, after anchoring all to a common permutation and scaling, solves a master linear least squares problem per mode to estimate the full mode loading matrices $(\mathbf{A}, \mathbf{B}, \mathbf{C})$.

where $\mathbf{\Pi}_p$ is a permutation matrix, and $\mathbf{\Lambda}_p$ is a diagonal scaling matrix with nonzero elements on its diagonal. Assume that the first two columns of each $\mathbf{U}_p$ (rows of $\mathbf{U}_p^T$) are common, and let $\bar{\mathbf{U}}$ denote this common part, and $\bar{\mathbf{A}}_p$ denote the first two rows of $\tilde{\mathbf{A}}_p$. We therefore have

$$\bar{\mathbf{A}}_p = \bar{\mathbf{U}}^T \mathbf{A} \mathbf{\Pi}_p \mathbf{\Lambda}_p.$$

Dividing each column of $\bar{\mathbf{A}}_p$ by the element of maximum modulus in that column, and denoting the resulting $2 \times F$ matrix $\hat{\mathbf{A}}_p$, we obtain

$$\hat{\mathbf{A}}_p = \bar{\mathbf{U}}^T \mathbf{A} \mathbf{\Lambda} \mathbf{\Pi}_p.$$

Notice that $\mathbf{\Lambda}$ does not affect the ratio of elements in each 2×1 column. If these ratios are distinct (which is guaranteed almost surely if $\bar{\mathbf{U}}$ and $\mathbf{A}$ are independently drawn from absolutely continuous distributions), then the different permutations can be matched by sorting the ratios of the two coordinates of each 2×1 column of $\hat{\mathbf{A}}_p$.

In practice using a few more "anchor" rows will improve the permutation-matching performance, and is recommended in difficult cases with high noise variance. When S anchor rows are used, the optimal permutation matching problem can be cast as

$$\min_{\mathbf{\Pi}} ||\hat{\mathbf{A}}_1 - \hat{\mathbf{A}}_p \mathbf{\Pi}||_F^2,$$

where optimization is over the set of permutation matrices. This may appear to be a hard combinatorial problem at first sight; but it is not. Using

$$||\hat{\mathbf{A}}_1 - \hat{\mathbf{A}}_p \mathbf{\Pi}||_F^2 = \mathrm{Tr}\left((\hat{\mathbf{A}}_1 - \hat{\mathbf{A}}_p \mathbf{\Pi})^T (\hat{\mathbf{A}}_1 - \hat{\mathbf{A}}_p \mathbf{\Pi})\right) =$$

$$||\hat{\mathbf{A}}_1||_F^2 + ||\hat{\mathbf{A}}_p\mathbf{\Pi}||_F^2 - 2\text{Tr}(\hat{\mathbf{A}}_1^T\hat{\mathbf{A}}_p\mathbf{\Pi}) =$$
$$||\hat{\mathbf{A}}_1||_F^2 + ||\hat{\mathbf{A}}_p||_F^2 - 2\text{Tr}(\hat{\mathbf{A}}_1^T\hat{\mathbf{A}}_p\mathbf{\Pi}).$$

It follows that we may instead

$$\max_{\mathbf{\Pi}} \text{Tr}(\hat{\mathbf{A}}_1^T\hat{\mathbf{A}}_p\mathbf{\Pi}),$$

over the set of permutation matrices. This is what is known as the *Linear Assignment Problem* (LAP), and it can be efficiently solved using the *Hungarian Algorithm*.

After this column permutation-matching process, we go back to (6.1) and permute its columns to obtain $\breve{\mathbf{A}}_p$ satisfying

$$\breve{\mathbf{A}}_p = \mathbf{U}_p^T\mathbf{A}\mathbf{\Pi}\mathbf{\Lambda}_p.$$

It remains to get rid of $\mathbf{\Lambda}_p$. For this, we normalize each column by dividing it with its norm. This finally yields

$$\breve{\mathbf{A}}_p = \mathbf{U}_p^T\mathbf{A}\mathbf{\Pi}\mathbf{\Lambda}.$$

For recovery of $\mathbf{A}$ up to permutation and scaling of its columns, we then require that the matrix of the linear system

$$\begin{bmatrix} \breve{\mathbf{A}}_1 \\ \vdots \\ \breve{\mathbf{A}}_P \end{bmatrix} = \begin{bmatrix} \mathbf{U}_1^T \\ \vdots \\ \mathbf{U}_P^T \end{bmatrix} \mathbf{A}\mathbf{\Pi}\mathbf{\Lambda} \tag{6.2}$$

be full column rank. This implies that

$$2 + \sum_{p=1}^{P}(L_p - 2) \geq I$$

i.e.,

$$\sum_{p=1}^{P}(L_p - 2) \geq I - 2.$$

Note that every sub-matrix contains the two anchor rows which are common, and duplicate rows clearly do not increase the rank. Also note that once the dimensionality requirement is met, the matrix will be full rank with probability 1, because its non-redundant entries are drawn from a jointly continuous distribution (by design).

Assuming $L_p = L, \forall p \in \{1, \cdots, P\}$ for simplicity (and symmetry of computational load), we obtain $P(L-2) \geq I - 2$, or, in terms of the number of threads

$$P \geq \frac{I-2}{L-2}.$$

Likewise, from the corresponding full column rank requirements for the other two modes, we obtain

$$P \geq \frac{J}{M}, \text{ and } P \geq \frac{K}{N}.$$

Notice that we do not subtract 2 from numerator and denominator for the other two modes, because the permutation of columns of $\tilde{\mathbf{A}}_p, \tilde{\mathbf{B}}_p, \tilde{\mathbf{C}}_p$ is *common* - so it is enough to figure it out from one mode, and apply it to other modes as well. In short,

$$P \geq \max\left(\frac{I-2}{L-2}, \frac{J}{M}, \frac{K}{N}\right)$$

Note that if, say, $\mathbf{A}$ can be identified and it is full column rank, then $\mathbf{B}$ and $\mathbf{C}$ can be identified by solving a linear least squares problem - but this requires access to the full big tensor data. In the same vein, if $\mathbf{A}$ and $\mathbf{B}$ are identified, then $\mathbf{C}$ can be identified from the full big tensor data even if $\mathbf{A}$ and $\mathbf{B}$ are not full column rank individually - it is enough that $\mathbf{A} \odot \mathbf{B}$ is full column rank, which is necessary for identifiability of $\mathbf{C}$ even from the big tensor, hence not restrictive. PARACOMP-based identification, on the other hand, only requires access to the factors derived from the small replicas. This is clearly advantageous, as the raw big tensor data can be discarded after compression, and there is no need for retrieving huge amounts of data from cloud storage.

We have thus established the following theorem:

Theorem 6.7.:
The data for each thread $\mathbf{y}_p := \text{vec}\left(\underline{\mathbf{Y}}_p\right)$ can be uniquely factored, i.e., $\left(\tilde{\mathbf{A}}_p, \tilde{\mathbf{B}}_p, \tilde{\mathbf{C}}_p\right)$ is unique up to column permutation and scaling. If, in addition to the above, we also have $P \geq \max\left(\frac{I}{L}, \frac{J}{M}, \frac{K-2}{N-2}\right)$ parallel threads, then $(\mathbf{A}, \mathbf{B}, \mathbf{C})$ are almost surely identifiable from the thread outputs $\left\{\left(\tilde{\mathbf{A}}_p, \tilde{\mathbf{B}}_p, \tilde{\mathbf{C}}_p\right)\right\}_{p=1}^{P}$ up to a common column permutation and scaling.

Proof. We can pick the mode used to figure out the permutation ambiguity, leading to the symmetrized condition $P \geq \min\{P_1, P_2, P_3\}$ with

$$P_1 = \max\left(\frac{I-2}{L-2}, \frac{J}{M}, \frac{K}{N}\right)$$

$$P_2 = \max\left(\frac{I}{L}, \frac{J-2}{M-2}, \frac{K}{N}\right)$$

$$P_3 = \max\left(\frac{I}{L}, \frac{J}{M}, \frac{K-2}{N-2}\right)$$

If the compression ratios in the different modes are similar, it makes sense to use the longest mode for this purpose; if this is the last mode, then

$$P \geq \max\left(\frac{I}{L}, \frac{J}{M}, \frac{K-2}{N-2}\right)$$

In reference to Fig. 6.2, assume $\mathbf{x} := \text{vec}\left(\underline{\mathbf{X}}\right) = (\mathbf{A} \odot \mathbf{B} \odot \mathbf{C})\,\mathbf{1} \in \mathbb{R}^{IJK}$, where $\mathbf{A}$ is $I \times F$, $\mathbf{B}$ is $J \times F$, $\mathbf{C}$ is $K \times F$ (i.e., the rank of $\underline{\mathbf{X}}$ is at most F). Assume that $F \leq I \leq J \leq K$, and $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ are all full column rank (F). Further assume that $L_p = L$, $M_p = M$, $N_p = N$,

$\forall p \in \{1, \cdots, P\}$, $L \leq M \leq N$, $(L+1)(M+1) \geq 16F$, the elements of $\{\mathbf{U}_p\}_{p=1}^{P}$ are drawn from a jointly continuous distribution, and likewise for $\{\mathbf{V}_p\}_{p=1}^{P}$, while each $\mathbf{W}_p$ contains two common anchor columns, and the elements of $\{\mathbf{W}_p\}_{p=1}^{P}$ (except for the repeated anchors, obviously) are drawn from a jointly continuous distribution. ■

The above result is indicative of a family of results that can be derived, using different PARAFAC identifiability results. Its significance may not be immediately obvious, so it is worth elaborating further at this point. On one hand, Theorem 6.7 shows that fully parallel computation of the big tensor decomposition is possible – the first such result, to the best of our knowledge, that *guarantees* identifiability of the big tensor decomposition from the intermediate small tensor decompositions, without placing stringent additional constraints. On the other hand, the conditions appear convoluted, and the memory / storage and computational savings, if any, are not necessarily easy to see. The following claim nails down the take-home message.

Claim 6.2.:
Under the conditions of Theorem 6.7, if $\frac{K-2}{N-2} = \max\left(\frac{I}{L}, \frac{J}{M}, \frac{K-2}{N-2}\right)$, then the memory / storage and computational complexity savings afforded by the architecture shown in Fig. 6.2 relative to brute-force computation are of order $\frac{IJ}{F}$.

Proof. Each thread must store LMN elements, and we have $P = \frac{K-2}{N-2}$ threads in all, leading to a total data size of order LMK versus IJK, so the ratio is $\frac{IJ}{LM}$. The condition $(L+1)(M+1) \geq 16F$ only requires LM to be of order F, hence the total compression ratio can be as high as $O\left(\frac{IJ}{F}\right)$. Turning to overall computational complexity, note that optimal low-rank tensor factorization is NP-hard, even in the rank-one case. Practical tensor factorization algorithms, however, typically have complexity $O(IJKF)$ (per iteration, and overall if a bound on the maximum number of iterations is enforced). It follows that the practical complexity order for factoring out the P parallel threads is $O(PLMNF)$ versus $O(IJKF)$ for the brute-force computation. Taking into account the lower bound on P, the ratio is again of order $\frac{IJ}{LM}$, and since the condition $(L+1)(M+1) \geq 16F$ only requires LM to be of order F, the total computational complexity gain can be as high as $O\left(\frac{IJ}{F}\right)$. ■

Remark 6.2.:
The complexity of solving the master linear equation (6.2) in the final merging step for $\mathbf{A}$ may be a source of concern - especially because it hasn't been accounted for in the overall complexity calculation. Solving a linear system of order of I equations in I unknowns generally requires $O(I^3)$ computations; but closer scrutiny of the system matrix in (6.2) reveals interesting features. If all elements of the compression matrices $\{\mathbf{U}_p\}$ (except for the common anchors) are independent and identically distributed with zero mean and unit variance, then, after removing the redundant rows, the system matrix in (6.2) will have approximately orthogonal columns for large I. This implies that its left pseudo-inverse will simply be its transpose, approximately. This reduces the complexity of solving (6.2) to I^2F. If higher accuracy is required, the pseudo-inverse may be computed off-line and stored. It is also important to stress that (6.2) is only solved once for each

mode at the end of the overall process, whereas tensor decomposition typically takes many iterations. In short, the constants are such that we need to worry more about the compression (fork) and decomposition stages, rather than the final join stage.

Finally, the following theorem demonstrates the conditions for identifiability of PARACOMP's result:

Theorem 6.8.:
In reference to Fig. 6.2, assume $\mathbf{x} := \text{vec}\left(\underline{\mathbf{X}}\right) = \left(\mathbf{A} \odot \mathbf{B} \odot \mathbf{C}\right) \mathbf{1} \in \mathbb{R}^{IJK}$, where $\mathbf{A}$ is $I \times F$, $\mathbf{B}$ is $J \times F$, $\mathbf{C}$ is $K \times F$ (i.e., the rank of $\underline{\mathbf{X}}$ is at most F). Assume that $I \geq F$, $J \geq F$ (K can be $< F$), and pick $L_p = L$, $M_p = M$, $N_p = N, \forall p \in \{1, \cdots, P\}$, with $L = M = F$, and $N = 3$. The compression matrices are chosen as in Theorem 6.7. If $P \geq \max\left(\frac{I}{L}, \frac{J}{M}, K-2\right)$, then $(\mathbf{A}, \mathbf{B}, \mathbf{C})$ is identifiable from $\left\{\left(\tilde{\mathbf{A}}_p, \tilde{\mathbf{B}}_p, \tilde{\mathbf{C}}_p\right)\right\}_{p=1}^{P}$, for almost every $(\mathbf{A}, \mathbf{B}, \mathbf{C})$ and almost every set of compression matrices. When $\frac{I}{L} = \max\left(\frac{I}{L}, \frac{J}{M}, K-2\right)$, the total storage and complexity gains are of order $\frac{JK}{F}$; whereas if $K-2 = \max\left(\frac{I}{L}, \frac{J}{M}, K-2\right)$, the total storage and complexity gains are of order $\frac{IJ}{F^2}$.

Proof. Theorem 6.7 assumes $F \leq \min(I, J, K)$ in order to ensure (via Lemma 6.2) absolute continuity of the compressed factor matrices, which is needed to invoke almost sure uniqueness per [CO12]. Cases where $F > \min(I, J, K)$ can be treated using Kruskal's condition for unique decomposition of each compressed replica

$$\min(L, k_{\mathbf{A}}) + \min(M, k_{\mathbf{B}}) + \min(N, k_{\mathbf{C}}) \geq 2F + 2.$$

It can be shown that $k_{\mathbf{A}} = \min(I, F)$ for almost every $\mathbf{A}$ (except for a set of measure zero in $\mathbb{R}^{IF}$); and likewise $k_{\mathbf{B}} = \min(J, F)$, and $k_{\mathbf{C}} = \min(K, F)$, for almost every $\mathbf{B}$ and $\mathbf{C}$. This simplifies[2] the above condition to

$$\min(L, I, F) + \min(M, J, F) + \min(N, K, F) \geq 2F + 2.$$

Assume $I \geq F$, $J \geq F$, but $K < F$, and pick $L = M = F$, and $N = 3$. Then the condition further reduces to

$$2F + \min(3, K) \geq 2F + 2,$$

which is satisfied for any $K \geq 2$ (i.e., for any tensor). We also need

$$P \geq \max\left(\frac{I}{L}, \frac{J}{M}, \frac{K-2}{N-2}\right),$$

which in this case ($N = 3$) reduces to

$$P \geq \max\left(\frac{I}{L}, \frac{J}{M}, K-2\right).$$

[2]Meaning: if the simplified condition holds, then PARAFAC decomposition of each reduced replica is unique for almost every $(\mathbf{A}, \mathbf{B}, \mathbf{C})$ and almost every set of compression matrices $(\mathbf{U}, \mathbf{V}, \mathbf{W})$.

When $\frac{I}{L} = \max\left(\frac{I}{L}, \frac{J}{M}, K-2\right)$, then there are $\frac{I}{L}$ parallel threads of size $LMN = 3F^2$ each, for total cloud storage $3IF$, i.e., order IF; hence the overall compression ratio (taking all replicas into account) is of order $\frac{IJK}{IF} = \frac{JK}{F}$. The ratio of overall complexity orders is also $\frac{IJKF}{IF^2} = \frac{JK}{F}$. This is the same type of result as the one we derived for the case $F \leq \min(I, J, K)$. On the other hand, when $K-2 = \max\left(\frac{I}{L}, \frac{J}{M}, K-2\right)$, there are $K-2$ parallel threads of size $LMN = 3F^2$ each, for total cloud storage $3F^2(K-2)$, i.e., order KF^2; hence the overall compression ratio is $\frac{IJK}{KF^2} = \frac{IJ}{F^2}$, and the ratio of overall complexity orders is also $\frac{IJKF}{KF^3} = \frac{IJ}{F^2}$. We see that there is a penalty factor F relative to the case $F \leq \min(I, J, K)$; this is likely an artifact of the method of proof, which we hope to improve in future work. ■

6.5.1 Latent sparsity

If latent sparsity is present, we can exploit it to reduce P. Assume that every column of $\mathbf{A}$ $(\mathbf{B}, \mathbf{C})$ has at most n_a (resp. n_b, n_c) nonzero elements. A column of $\mathbf{A}$ can be uniquely recovered from only $2n_a$ incoherent linear equations [DE03]. Therefore, we may replace the condition

$$P \geq \max\left(\frac{I}{L}, \frac{J}{M}, \frac{K-2}{N-2}\right),$$

with

$$P \geq \max\left(\frac{2n_a}{L}, \frac{2n_b}{M}, \frac{2n_c-2}{N-2}\right). \tag{6.3}$$

Assuming

$$\frac{2n_c-2}{N-2} = \max\left(\frac{2n_a}{L}, \frac{2n_b}{M}, \frac{2n_c-2}{N-2}\right),$$

it is easy to see that the total cloud storage and complexity gains are of order $\frac{IJ}{F}\frac{K}{n_c}$ - improved by a factor of $\frac{K}{n_c}$. It is interesting to compare this result with the one in Sidiropoulos & Kyrillidis [SK12], which corresponds to using $P = 1$ in our present context. Notice that (6.3) implies $L \geq \frac{2n_a}{P}$, $M \geq \frac{2n_b}{P}$, $N-2 \geq \frac{2n_c-2}{P} \Rightarrow N \geq \frac{2n_c}{P} + 2(1-\frac{1}{P})$ $\Rightarrow N \geq \frac{2n_c}{P}$. Substituting $P = 1$ we obtain $L \geq 2n_a$, $M \geq 2n_b$, $N \geq 2n_c$, which is exactly the condition required in [SK12]. We see that PARACOMP subsumes [SK12], offering greater flexibility in terms of choosing P to reduce the size of replicas for easier in-memory processing, at the cost of an additional merging step at the end. Also note that PARACOMP is applicable in the case of dense latent factors, whereas [SK12] is not.

Remark 6.3.:
In practice we will use a higher P, i.e.,

$$P \geq \max\left(\frac{\mu n_a}{L}, \frac{\mu n_b}{M}, \frac{\mu n_c - 2}{N-2}\right),$$

with $\mu \in \{3, 4, 5\}$ instead of 2, and an ℓ_1 sparse under-determined linear equations solver for the final merging step for $\mathbf{A}$. This will increase complexity from $O(I^2F)$ to $O(I^{3.5}F)$, and the constants are such that the difference is significant. This is the price paid for the reduced memory and intermediate complexity benefits afforded by latent sparsity.

6.6 Experimental Evaluation

Our theorems ensure that PARACOMP works with ideal low- and known-rank tensors, but what if there is measurement noise or other imperfections, or we underestimate the rank? Does the overall approach fall apart in this case? In this section, we provide indicative results to illustrate what can be expected from PARACOMP and the effect of various parameters on estimation performance.

In all cases considered, $I = J = K = 500$, the noiseless tensor has rank $F = 5$, and is synthesized by randomly and independently drawing $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, each from an i.i.d. zero-mean, unit-variance Gaussian distribution (`randn(500,5)` in Matlab), and then taking their tensor product; i.e., computing the sum of outer products of corresponding columns of $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$. Gaussian i.i.d. measurement noise is then added to this noiseless tensor to yield the observed tensor to be analyzed. The nominal setup uses $L = M = N = 50$ (so that each replica is 0.1% of the original tensor), and $P = 12$ replicas are created for the analysis (so the overall cloud storage used for all replicas is 1.2% of the space needed to store the original tensor). $S = 3$ common anchor rows (instead of $S = 2$, which is the minimum possible) are used to fix the permutation and scaling ambiguity. These parameter choices satisfy PARACOMP identifiability conditions without much additional 'slack'. The standard deviation of the measurement noise is nominally set to $\sigma = 0.01$.

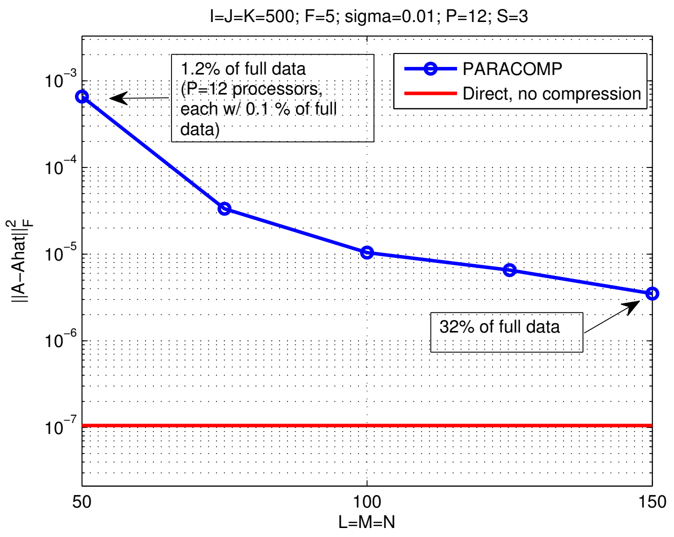


Figure 6.3: **PARACOMP is accurate**: MSE as a function of $L = M = N$.

Fig. 6.3 shows the total squared error for estimating $\mathbf{A}$, i.e., $||\mathbf{A} - \hat{\mathbf{A}}||_2^2$, where $\hat{\mathbf{A}}$ denotes the estimate of $\mathbf{A}$ obtained using PARACOMP, as a function of $L = M = N$. The baseline is the total squared error attained by directly fitting the uncompressed $500 \times 500 \times 500$ tensor using the ALS algorithm (as we saw in Section 3.2.1.2 of Chapter 2) - the size of the uncompressed tensor used here makes such direct fitting possible, for comparison purposes. We see that PARACOMP yields respectable accuracy with

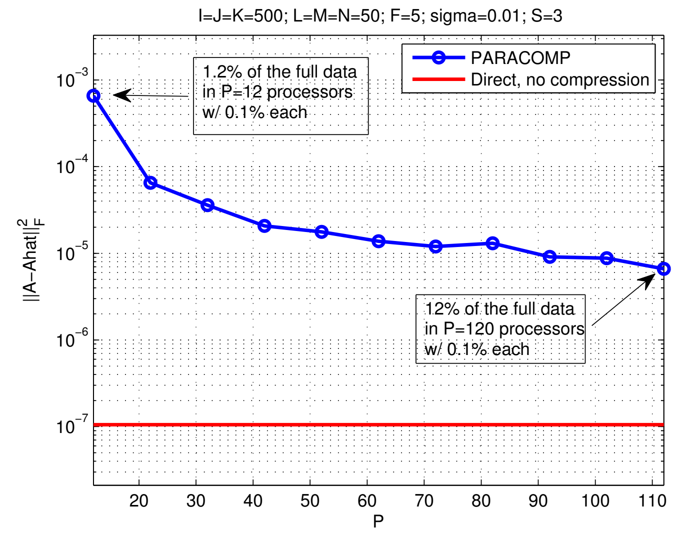


Figure 6.4: **Increasing replicas improves PARACOMP accuracy**: MSE as a function of P, the number of replicas / parallel threads spawned.

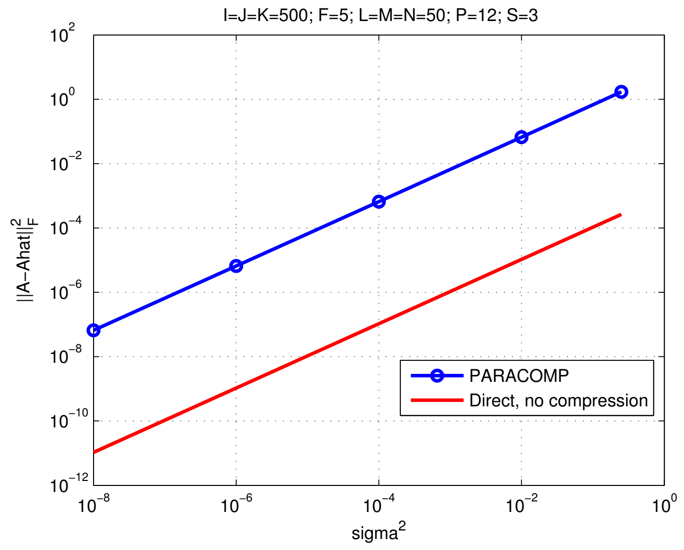


Figure 6.5: **PARACOMP works for well for noisy data**: MSE as a function of additive white Gaussian noise variance σ^2.

only 1.2% of the full data, and is just an order of magnitude worse than the baseline algorithm when $L = M = N = 150$, corresponding to 32% of the full data. This is one way we can trade-off memory/storage/computation versus estimation accuracy in the PARACOMP framework: by controlling the size of each replica. Another way to trade-off memory/storage/computation for accuracy is through P. Fig. 6.4 shows accuracy as a function of the number of replicas (computation threads) P, for fixed $L = M = N = 50$. Finally, Fig. 56.5 plots accuracy as a function of measurement noise variance σ^2, for

$L = M = N = 50$ and $P = 12$.

6.7 Conclusions

We have reviewed the basics of tensors and tensor decomposition, and presented a novel architecture for parallel and distributed computation of low-rank tensor decomposition that is especially well-suited for big tensors. It is based on parallel processing of a set of randomly compressed, reduced-size 'replicas' of the big tensor. Our main contributions are:

- **Efficient Algorithm**: PARACOMP affords memory / storage and complexity gains of order up to $\frac{IJ}{F}$ for a big tensor of size $I \times J \times K$ of rank F. No sparsity is required, although such sparsity can be exploited to improve memory, storage and computational savings.
- **Flexible Scalable Architecture**: Each replica is independently decomposed, and the results are joined via a master linear equation per tensor mode. The number of replicas and the size of each replica can be adjusted to fit the number of computing nodes and the memory available to each node, and each node can run its own PARAFAC software, depending on its computational capabilities. This flexibility is why PARACOMP is better classified as a computational architecture, as opposed to a method or algorithm.
- **Identifiability guarantees**: PARACOMP enables massive parallelism with guaranteed identifiability properties: if the big tensor is indeed of low rank and the system parameters are appropriately chosen, then the rank-one factors of the big tensor will indeed be recovered from the analysis of the reduced-size replicas.

Part II

Algorithms - Unsupervised Quality Assessment

Chapter based on material published in [PF15].

Chapter 7

Fast and Scalable Core Consistency Diagnostic for the PARAFAC Decomposition for Big Sparse Tensors

Judging the quality of the PARAFAC decomposition for two orders of magnitude larger data.

The Core Consistency Diagnostic (CORCONDIA) (see also Section 3.2.8 of Chapter 3) is a very intuitive and simple heuristic for determining the quality of PARAFAC decomposition. However simple, computation of this diagnostic proves to be a very daunting task even for data of medium scale. In this Chapter we derive a fast and *exact* algorithm for CORCONDIA which exploits data sparsity and scales very well as the tensor size increases. Our algorithm operates on data that are at least 100 times larger than what the state-of-the-art can handle.

7.1 Introduction

Multilinear analysis and tensor decompositions have been increasingly popular in a very wide variety of fields, ranging from signal processing to data mining.

The majority of existing applications of tensor decompositions in fields such as Chemometrics, focus on dense data, where most of the values of a tensor are observed and non-zero [Bro97]. However, recently, there has emerged an increasing interest in applying tensor decompositions on sparse data, where most of the coefficients of the tensor are unobserved; examples of such data can be links between web-pages [KB06, KS08], computer networks [KS08, MGF11, PFS12], Knowledge Base data [KPHF12, PFS12], citation networks [KS08] and social networks [BHK06, KS08, PFS12].

As a running motivating example we choose that of social networks. Consider an online social network platform such as Facebook, which records relations and interactions between its users. Throughout the vast amount of Facebook users (estimated to be around 1.3 Billion), it is physically impossible for all users to interact with each other; i.e. a certain user interacts with a very small fraction of the total number of users. Thus, suppose that we observe a tensor of user interactions over time (i.e. the modes are (user, user, time)), the number of non-zero values of this tensor will be very small, resulting in a highly sparse tensor with very high dimensions.

There are many challenges posed by the aforementioned type of tensors. In particular, traditional *dense* methods that assume that all data can (and should) be stored in main memory fall short. Computation of tensor decompositions in such scenarios was pioneered by Kolda and Bader in [KB06] and [BK07], where they introduce efficient in-memory computations for sparse tensors, taking advantage of the sparse structure and avoiding storing the entire data into memory. Subsequently, in [KPHF12], the authors apply this principle in a distributed *cloud* setting, where a tensor can span terabytes of storage.

Coming back to the example of the large sparse tensor that represents the time-evolving social network, the rank of its PARAFAC decomposition [Har70], i.e. the number of rank-one factors that we extract from the data, reflects the near cliques or communities in that network [PAI13]. Thus, it is important to be able to determine the decomposition rank efficiently. Unfortunately, even determining the true rank of a tensor, contrary to the matrix case (where the solution is given to us by the Singular Value Decomposition, in polynomial time), is an NP-hard problem [HL13]. Fortunately, however, there exist heuristic methods which, given an R rank decomposition, are able to provide a *diagnostic* that captures how well this decomposition models the tensor data. The first such heuristic, CORCONDIA, appeared in [Bro98] and subsequently described in detail in [BK03]; there exist more recent approaches which further extend the main idea of CORCONDIA [dCHR08].

The state of the art algorithms for CORCONDIA, so far have focused on dense and relatively small datasets; however, when we shift our attention to data of much larger scale (such as the running example of the time-evolving social network), state of the art approaches suffer, understandably so, from scalability issues. In this chapter, we make such diagnostics available for the analysis of very large tensors. In particular, our contributions are

- We derive an equivalent formula for computing CORCONDIA, and propose an efficient algorithm, able to scale on very high dimensional sparse tensors.
- We apply our algorithm to a big real-world time-evolving social network dataset,

demonstrating its practicality in the analysis of big tensor/graph data.

- We make our code publicly available at http://www.cs.cmu.edu/~epapalex/src/efficient_corcondia.zip

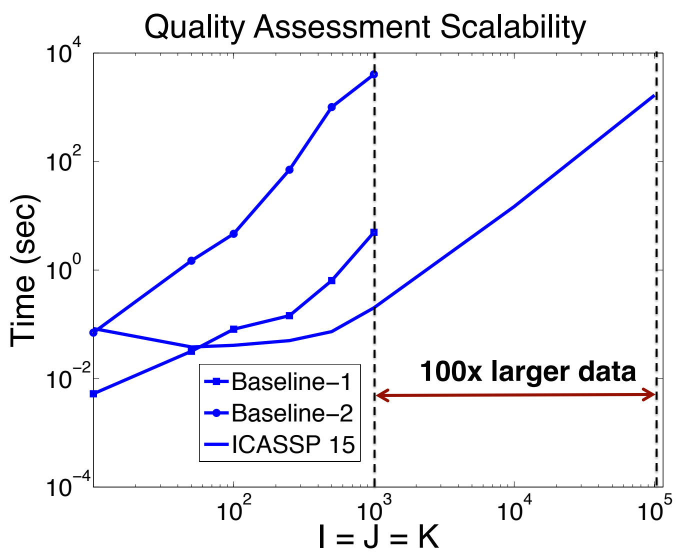


Figure 7.1: Our algorithm is able to analyze the quality for at least 100 times larger tensors of dimensions $I \times I \times I$ and I non-zeros.

A snapshot of our results is shown in Fig. 7.1, where our algorithm was able to run on 100 times larger data than the state-of-the-art for sparse tensors.

7.2 Background & Problem Formulation

Here we repeat necessary notation and definitions that also appear in Chapter 2.

7.2.1 A Note on Notation

Notation used here follows Table 2.1. A tensor is denoted by $\underline{\mathbf{X}}$. A matrix is denoted by $\mathbf{X}$. A vector is denoted by $\mathbf{x}$. The symbol $\circ$ denotes the outer product. The symbol $\otimes$ denotes the Kronecker product. The symbol $\dagger$ denotes the Moore-Penrose pseudoinverse. The symbol $vec\left(\right)$ denotes the vectorization operation. Finally, the operation of a series of Kronecker products times a vector, i.e., $\left(\mathbf{A}_1 \otimes \mathbf{A}_2 \otimes \cdots \otimes \mathbf{A}_k\right)\mathbf{x}$ is denoted by $\textsc{KronMatVec}\left(\{\mathbf{A}_1 \otimes \mathbf{A}_2 \otimes \cdots \otimes \mathbf{A}_N\}, \mathbf{x}\right)$.

7.2.2 Brief Introduction to Tensor Decompositions

Given a $I \times J \times K$ tensor $\underline{\mathbf{X}}$, we can decompose it according to the PARAFAC decomposition [Har70] as a sum of rank-one tensors:

$$\underline{\mathbf{X}} \approx \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r$$

where the (i,j,k) entry of $\mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r$ is $\mathbf{a}_r(i)\mathbf{b}_r(j)\mathbf{c}_r(k)$. Usually, PARAFAC is represented in its matrix form $[\mathbf{A}, \mathbf{B}, \mathbf{C}]$, where the columns of matrix $\mathbf{A}$ are the $\mathbf{a}_r$ vectors (and accordingly for $\mathbf{B}, \mathbf{C}$). The PARAFAC decomposition is especially useful when we are interested in extracting the true latent factors that generate the tensor.

Another very popular Tensor decomposition is the TUCKER model [KDL80], where a tensor is decomposed into rank-one factors times a core tensor:

$$\underline{\mathbf{X}} \approx \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} \underline{\mathbf{G}}(p,q,r)\mathbf{u}_p \circ \mathbf{v}_q \circ \mathbf{w}_r$$

where $\mathbf{U}, \mathbf{V}, \mathbf{W}$ are orthogonal. The TUCKER model is especially used for compression. Furthermore, PARAFAC can be seen as a restricted TUCKER model, where the core tensor $\underline{\mathbf{G}}$ is super-diagonal, i.e. non-zero values are only in the entries where $i = j = k$. This observation will be useful in order to motivate the CORCONDIA diagnostic.

7.2.3 Brief Introduction to CORCONDIA

As outlined in the Introduction and Chapter 3, there exist a few diagnostics/heuristics for assessing the modelling quality of the PARAFAC decomposition. In this work, we will focus on CORCONDIA [Bro98, BK03], which is the simplest and most intuitive to describe. However, [dCHR08] which builds upon CORCONDIA can also benefit from our proposed algorithm.

In a nutshell, the idea behind CORCONDIA is the following: Given a tensor $\underline{\mathbf{X}}$ and its PARAFAC decomposition $\mathbf{A}, \mathbf{B}, \mathbf{C}$, one could imagine fitting a TUCKER model where matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$ are the loading matrices of the TUCKER model and $\underline{\mathbf{G}}$ is the core tensor (which we need to solve for). Since, as we already mentioned, PARAFAC can be seen as a restricted TUCKER model with super-diagonal core tensor, if our PARAFAC modelling of $\underline{\mathbf{X}}$ using $\mathbf{A}, \mathbf{B}, \mathbf{C}$ is good, core tensor $\underline{\mathbf{G}}$ should be as close to super-diagonal as possible. If there are deviations from the super-diagonal, then this is a good indication that our PARAFAC model is somehow flawed (either the decomposition rank is not appropriate, or the data do not have the appropriate structure).

As it is highlighted in [Bro98], since matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$ are not orthogonal, we may not use typical algorithms that are used to fit the TUCKER model (e.g page 72 of [Bro98]). Instead, we can pose the problem as the following least squares problem:

$$\min_{\underline{\mathbf{G}}} \|vec\left(\underline{\mathbf{X}}\right) - \left(\mathbf{A} \otimes \mathbf{B} \otimes \mathbf{C}\right) vec\left(\underline{\mathbf{G}}\right)\|_F^2$$

with solution: $vec\left(\underline{\mathbf{G}}\right) = \left(\mathbf{A} \otimes \mathbf{B} \otimes \mathbf{C}\right)^{\dagger} vec\left(\underline{\mathbf{X}}\right)$

7.3 Problem Definition & Proposed Method

Albeit simple and elegant, the solution of the Least Squares problem that lies in the heart of CORCONDIA suffers in the case of high dimensional data. In particular, this straightforward solution requires to first compute and store $(\mathbf{A} \otimes \mathbf{B} \otimes \mathbf{C})$ and then pseudoinvert it. Consider a $10^4 \times 10^4 \times 10^4$ tensor; even for a very low rank decomposition of $R = 10$, the aforementioned Kronecker product will be of size $10^{12} \times 10^3$, a fact which renders computing and storing such a matrix highly impractical (if not outright impossible), and subsequently, computing its pseudoinverse largely intractable. Even if the factor matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$ are sparse [PSB13] (resulting in a sparse Kronecker product), pseudoinverting a matrix of such large dimensions is very computationally challenging.

In this section, we describe our proposed algorithm. Our "wish-list" of properties for our algorithm is the following:

1. Avoid materializing any Kronecker product.
2. Avoid directly pseudo-inverting the (potentially huge) aforementioned Kronecker product.
3. Exploit any sparse structure in the factor matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$ and/or the tensor $\underline{\mathbf{X}}$.

In order to achieve the above, we need to reformulate the computation of CORCONDIA.

Proposition 7.1.:
The pseudoinverse $(\mathbf{A} \otimes \mathbf{B} \otimes \mathbf{C})^\dagger$ can be rewritten as

$$\left(\mathbf{V_a} \otimes \mathbf{V_b} \otimes \mathbf{V_c}\right)\left(\mathbf{\Sigma_a}^{-1} \otimes \mathbf{\Sigma_b}^{-1} \otimes \mathbf{\Sigma_c}^{-1}\right)\left(\mathbf{U_a}^T \otimes \mathbf{U_b}^T \otimes \mathbf{U_c}^T\right)$$

where $\mathbf{A} = \mathbf{U_a}\mathbf{\Sigma_a}\mathbf{V_a}^\mathbf{T}$, $\mathbf{B} = \mathbf{U_b}\mathbf{\Sigma_b}\mathbf{V_b}^\mathbf{T}$, and $\mathbf{C} = \mathbf{U_c}\mathbf{\Sigma_c}\mathbf{V_c}^\mathbf{T}$ (i.e. the respective Singular Value Decompositions).

Proof. For compactness, we show the two matrix case, but the extension to three matrices is straightforward. Using basic properties of the Kronecker product from [Neu69], and rewriting $\mathbf{A}, \mathbf{B}$ using their SVD, we may write

$$\begin{aligned}(\mathbf{A} \otimes \mathbf{B}) &= \left[\left(\mathbf{U_a}\mathbf{\Sigma_a}\mathbf{V_a}^T\right) \otimes \left(\mathbf{U_b}\mathbf{\Sigma_b}\mathbf{V_b}^T\right)\right] \\ &= \left[\left(\mathbf{U_a}\mathbf{\Sigma_a}\right) \otimes \left(\mathbf{U_b}\mathbf{\Sigma_b}\right)\left(\mathbf{V_a} \otimes \mathbf{V_b}\right)^T\right] \\ &= \left[\left(\mathbf{U_a} \otimes \mathbf{U_b}\right)\left(\mathbf{\Sigma_a} \otimes \mathbf{\Sigma_b}\right)\left(\mathbf{V_a} \otimes \mathbf{V_b}\right)^T\right]\end{aligned}$$

By invoking properties shown in [Loa00], we can show that show that $(\mathbf{U_a} \otimes \mathbf{U_b})$ is orthonormal and $(\mathbf{\Sigma_a} \otimes \mathbf{\Sigma_b})$ is diagonal with non-negative entries.

Then,because the SVD is unique, then

$$\mathbf{A} \otimes \mathbf{B} = \left[\left(\mathbf{U_a} \otimes \mathbf{U_b}\right)\left(\mathbf{\Sigma_a} \otimes \mathbf{\Sigma_b}\right)\left(\mathbf{V_a} \otimes \mathbf{V_b}\right)^T\right]$$

is the SVD of $\mathbf{A} \otimes \mathbf{B}$.

Since the above is the SVD, then the Moore-Penrose pseudoinverse will be

$$\left(\mathbf{V_a} \otimes \mathbf{V_b}\right)\left({\mathbf{\Sigma_a}}^{-1} \otimes {\mathbf{\Sigma_b}}^{-1}\right)\left({\mathbf{U_a}}^T \otimes {\mathbf{U_b}}^T\right)$$

■

In [FF94] the authors first show an equivalent formulation of the Kronecker product pseudoinverse, which is however not expressed via the SVD, and thus suffers from numerical instabilities. Subsequently, they introduce a more complicated reformulation of the pseudoinverse of Kronecker products, where SVD is applied after computing the QR decomposition of the matrices involved. In this work we prefer the reformulation of Claim 1 since it is usually more efficient to compute in our case. In order to substantiate this claim we conducted the following experiment: Using Matlab (which to the best of our knowledge is the state of the art when it comes to efficient dense and sparse matrix computations in main memory), we computed the QR decomposition and the *economy* version of the SVD for a series of random matrices with increasing number of rows and fixed number of columns (a scenario which resembles the case where we have a tensor of increasing dimensions but we have a fixed rank). Figure 7.2 indicates the efficiency of SVD.

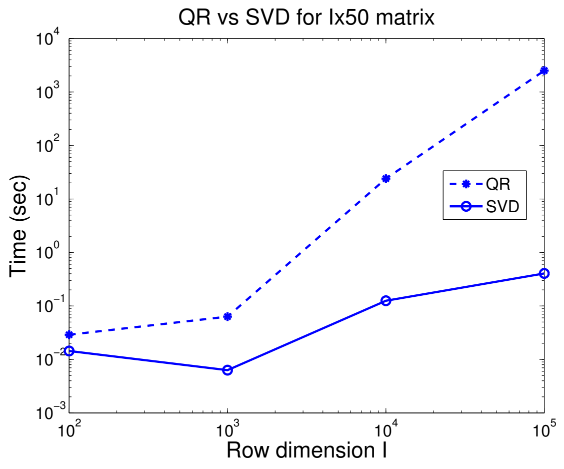


Figure 7.2: QR vs. SVD

It is important to note here that in the case that the factors $\mathbf{A}, \mathbf{B}, \mathbf{C}$ are sparse, we can exploit that fact using SVD for sparse matrices, to further speed up the computation. The straightforward algorithm that computes $(\mathbf{A} \otimes \mathbf{B} \otimes \mathbf{C})$ cannot take advantage of factor sparsity. Now, we can solve CORCONDIA without materializing any of the Kronecker products. Instead we observe that the equation we need to solve is

$$vec\left(\underline{\mathbf{G}}\right) = \left(\mathbf{V_a} \otimes \mathbf{V_b} \otimes \mathbf{V_c}\right)\left({\mathbf{\Sigma_a}}^{-1} \otimes {\mathbf{\Sigma_b}}^{-1} \otimes {\mathbf{\Sigma_c}}^{-1}\right)\left({\mathbf{U_a}}^T \otimes {\mathbf{U_b}}^T \otimes {\mathbf{U_c}}^T\right) vec\left(\underline{\mathbf{X}}\right)$$

which is a KRONMATVEC operation. $\mathbf{e}$, The fact that we transformed the operation to an equivalent one that contains a KRONMATVEC computation is extremely important, because KRONMATVEC can be carried out very efficiently, on-the-fly. In fact, [BD96] provide an efficient algorithm that does not materialize the (potentially very large) Kronecker product, and is able to efficiently compute KRONMATVEC $(\{\mathbf{A}_1 \otimes \mathbf{A}_2 \otimes \cdots \otimes \mathbf{A}_k\})\, \mathbf{x}$.

We use the algorithm of [BD96] in our proposed Algorithm 7.1. When the tensor is sparse, the above computations can be carried out very efficiently, using sparse matrix multiplication, resulting in huge gains in scalability, as we will show in the next Section.

Algorithm 7.1: Efficient CORCONDIA

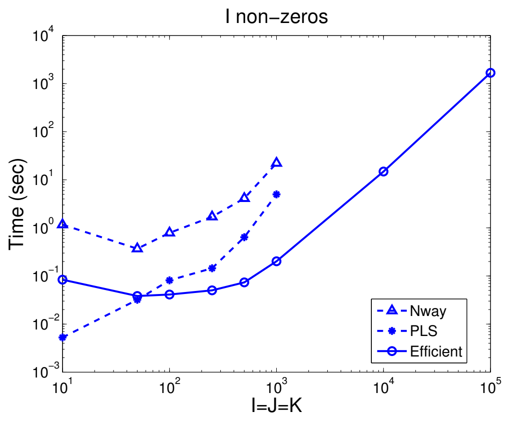
```
Input: Tensor $\underline{\mathbf{X}}$ and PARAFAC factor matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$.
Output: CORCONDIA diagnostic $c$.
1: Compute $\mathbf{A} = \mathbf{U_a}\mathbf{\Sigma_a}\mathbf{V_a}^T$
2: Compute $\mathbf{B} = \mathbf{U_b}\mathbf{\Sigma_b}\mathbf{V_b}^T$
3: Compute $\mathbf{C} = \mathbf{U_c}\mathbf{\Sigma_c}\mathbf{V_c}^T$
4: Calculate $\mathbf{y} = \left(\mathbf{U_a}^T \otimes \mathbf{U_b}^T \otimes \mathbf{U_c}^T\right) vec\left(\underline{\mathbf{X}}\right)$ using Algorithm of [BD96].
5: Calculate $\mathbf{z} = \left(\mathbf{\Sigma_a}^{-1} \otimes \mathbf{\Sigma_b}^{-1} \otimes \mathbf{\Sigma_c}^{-1}\right) \mathbf{y}$ using Algorithm of [BD96].
6: Calculate $vec\left(\underline{\mathbf{G}}\right) = \left(\mathbf{V_a} \otimes \mathbf{V_b} \otimes \mathbf{V_c}\right) \mathbf{z}$ using Algorithm of [BD96].
7: $c = 100\left(1 - \frac{\sum_{i=1}^{F}\sum_{j=1}^{F}\sum_{k=1}^{F}(\underline{\mathbf{G}}(i,j,k)-\underline{\mathbf{I}}(i,j,k))^2}{F}\right)$ (where $\underline{\mathbf{I}}$ is a super-diagonal tensor with ones
   on the $(i,i,i)$ entries).
```

7.4 Experimental Evaluation

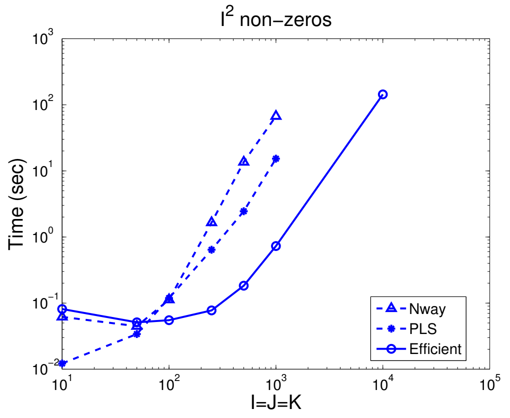
We implemented Algorithm 7.1 in Matlab, using the Tensor Toolbox [BK+15] (available at: http://www.sandia.gov/~tgkolda/TensorToolbox/index-2.5.html), which provides efficient manipulation and storage of sparse tensors. We make our code publicly available[1]. For comparisons, we used two baselines: the implementation of the N-way Toolbox for Matlab[AB00] (available at: http://www.models.life.ku.dk/nwaytoolbox), and the implementation of the PLS Toolbox (available at: http://www.eigenvector.com/software/pls_toolbox.htm). The PLS Toolbox is commercial, and is considered the state of the art for computing CORCONDIA, however, we include comparisons with the N-way Toolbox since it is freely available. All experiments were run on a workstation with 4 Intel(R) Xeon(R) E7- 8837 and 1TB of RAM.

Figure 7.3 shows the execution time as a function of the tensor mode dimension I for $I \times I \times I$ tensors for three cases: (a) very sparse tensors (with I non-zero values), (b) moderately sparse (with I^2 non-zero values), and (c) fully dense (with I^3 non-zero values). In Fig. 7.3(a), we see that our proposed algorithm is generally much faster than both baselines (note that the figures are in log-scales), while it keeps working for $I = 10^4$ where the baselines run out of memory. However, our proposed algorithm is able to scale up to $10^5 \times 10^5 \times 10^5$ tensors. In Fig. 7.3(b) we observe similar behavior, where

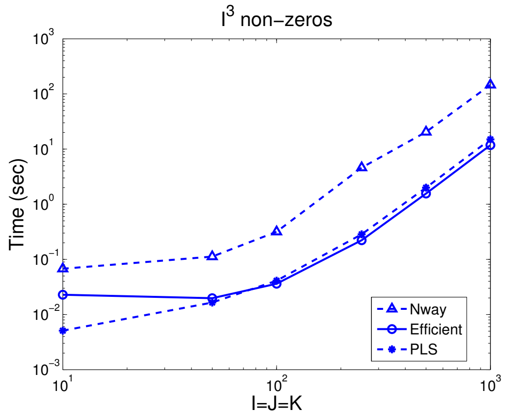
[1]Download our code at http://www.cs.cmu.edu/~epapalex/src/efficient_corcondia.zip



(a) Very Sparse



(b) Sparse



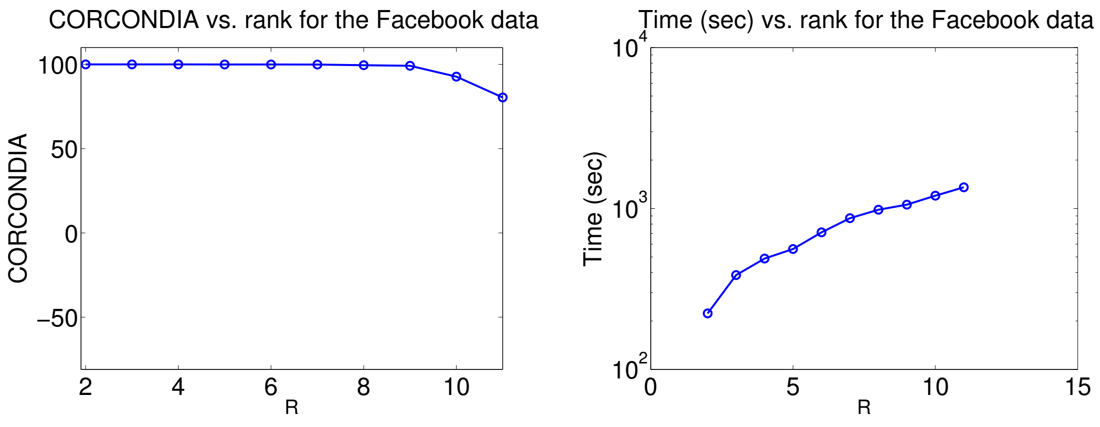
(c) Dense

Figure 7.3: **Our proposed method scales very well on sparse data**: This Figure shows time (sec) vs $I = J = K$, and for varying number of non-zeros. We see that especially in the very sparse case, our method is able to run on at least 100 times larger tensors compared to the two baselines.

the overall performance (for all three algorithms) is slightly lower, due to the existence of more non-zero values. Finally, in Fig. 7.3(c), as expected, we see that our proposed algorithm has the same performance as the state of the art PLS Toolbox implementation, since there is no sparsity to take advantage of.

7.5 Results on Real Data

In this section, we use our proposed algorithm in order to determine the appropriate number of components for the PARAFAC decomposition of a big real-world dataset. In particular, we chose a time-evolving social network dataset, more specifically a snapshot of Facebook (available at: `http://socialnetworks.mpi-sws.org/data-wosn2009.html`). This dataset records which user posted on another user's Wall and what date, forming a three-mode (User, User, Date) tensor, with dimensions: $63891 \times 63890 \times 1847$ and 737778 non-zero values. Despite the very large dimensions of the tensor, our algorithm is able to compute the CORCONDIA diagnostic for different values of the rank, as shown in Fig. 7.4(a). For each rank, we compute CORCONDIA 100 times and we show the maximum value attained, which reflects the best possible decomposition among those 100 iterations. In Fig. 7.4(b) we show the average time it took to compute the diagnostic for each rank.



(a) CORCONDIA vs. Rank (b) Time (sec) vs. Rank

Figure 7.4: Analyzing the Facebook tensor

7.6 Conclusions

In this chapter we propose an efficient algorithm for computing the CORCONDIA diagnostic, especially for large sparse tensors. The important take-home point is that in cases where either the tensor or the factors or both are sparse, our proposed algorithm significantly outperforms the state of the art baselines and scales very well as the data size grows. In the fully dense scenario, our proposed algorithm is as good as the state of the art.

Chapter based on material published in [Pap16].

Chapter 8

Automatic Tensor Mining with Unsupervised Quality Assessment

Automating PARAFAC in a data driven and unuspervised way.

How can we automatically decide how many PARAFAC components to extract, without having any ground truth? In this Chapter, building upon the results of Chapter 7 we introduce AUTOTEN, a novel automatic unsupervised tensor mining algorithm with minimal user intervention. We extensively evaluate AUTOTEN's performance on synthetic and real data, outperforming existing baselines.

8.1 Introduction

Tensor decompositions and their applications in mining multi-aspect datasets are ubiquitous and ever increasing in popularity. Data Mining application of these techniques has been largely pioneered by [KBK05] where the authors introduce a topical aspect to a graph between webpages, and extend the popular HITS algorithm in that scenario. Henceforth, the field of multi-aspect/tensor data mining has witnessed rich growth with prime examples of applications being social networks [KS08, LSC$^+$09, JCW$^+$14], citation networks [KS08], and computer networks [KS08], to name a few.

Tensor decompositions are undoubtedly a very powerful analytical tool with a rich variety of applications. However there exist research challenges in the field of data mining that need to be addressed, in order for tensor decompositions to claim their position as a de-facto tool for practitioners.

One challenge, which has received considerable attention, is the one of making tensor decompositions scalable to today's web scale. However, recently and especially for sparse tensors, there has been a substantial body of work with the first pioneering step by Kolda et al. [KBK05, BK07] exploiting sparsity for scalability; subsequently there have been distributed approaches based on the latter formulation [KPHF12], and other scalable approaches [PFS12, EM13]. By no means do we claim that scalability is a solved problem, however, we point out that there has been significant attention to it.

The main focus of this work, however, is on another, relatively less explored territory; that of assessing the *quality* of a tensor decomposition. In a great portion of tensor data mining, the task is exploratory and unsupervised: we are given a dataset, usually without any sort of ground truth, and we seek to extract *interesting* patterns or concepts from the data. It is crucial, therefore, to know whether a pattern that we extract actually models the data at hand, or whether it is merely modelling noise in the data. Especially in the age of Big Data, where feature spaces can be vast, it is imperative to have a measure of quality and avoid interpreting noisy, random variation that always exists in the data. Determining the number of components in a tensor is a very hard problem [HL13]. This is why, many seminal exploratory tensor mining papers, understandably, set the number of components manually [KBK05, STF06, KS08]. When there is a specific task at hand, e.g. link prediction [DKA11], recommendation [RST10], and supervised learning [HKP^{+}14], that entails some measure of success, then we can use cross-validation for selecting a good number of latent components which unfortunately cannot generalize to the case where labels or ground truth are absent.

However, not all hope is lost. There exists highly influential work in the Chemometrics literature [BK03] that introduces heuristics for determining the quality of a decomposition, taking into account properties of the PARAFAC decomposition [Har70] and being *application independent*, requiring no prior knowledge about the data. Inspired by and drawing from [BK03], we provide a comprehensive method for mining large and potentially sparse multi-aspect datasets using tensor decompositions.

Our contributions are:

- **Technology Transfer** To the best of our knowledge, this is the first data mining work that employs the Core Consistency Diagnostic for the quality assessment of a tensor decomposition; our sincere hope is to popularize such approaches in the data mining community, conducting a technology transfer from the Chemometrics community.
- **Algorithms** We propose AUTOTEN, a comprehensive, parallelizable methodology on mining multi-aspect datasets using tensors, which minimizes manual trial-and-error intervention and provides quality characterization of the solution (Section 8.3.2). Furthermore, we extend the Core Consistency Diagnostic of [BK03] assuming KL-divergence loss, which is more effective in modelling highly sparse, count data [CK12] (Section 8.3.1).
- **Evaluation & Discovery** We conduct a large scale study on 10 real datasets, exploring the structure of hidden patterns within these datasets (Section 8.5.1). To the

best of our knowledge, this is the first such broad study. As a data mining case study, we use AUTOTEN on real data discovering meaningful patterns (Section 8.5.2). Finally, we extensively evaluate our proposed method on synthetic data (Section 8.4).

In order to encourage **reproducibility**, most of the datasets used are public, and we make our code publicly available at `http://www.cs.cmu.edu/~epapalex/src/AutoTen.zip`.

8.2 Background

The notation used in this Chapter follows the general notation introduced in Chapter 2. Table 10.1 provides an overview of the symbols used in this Chapter in particular, some of which are specific to this Chapter.

Symbol	**Definition**
$\underline{\mathbf{X}}, \mathbf{X}, \mathbf{x}, x$	Tensor, matrix, column vector, scalar
$\circ$	Outer product
$vec(\,)$	Vectorization operator
$\otimes$	Kronecker product
$*\ \oslash$	Element-wise multiplication and division
$\mathbf{A}^\dagger$	Moore-Penrose Pseudoinverse of $\mathbf{A}$
$D_{KL}(a\|\|b)$	KL-Divergence
$\|\mathbf{A}\|_F$	Frobenius norm
KRONMATVEC	Efficient computation of $\mathbf{y} = (\mathbf{A}_1 \otimes \mathbf{A}_2 \otimes \cdots \otimes \mathbf{A}_n)\, \mathbf{x}$ [BD96]
$\mathbf{x}(i)$	i-th entry of $\mathbf{x}$ (same for matrices and tensors)
$\mathbf{X}(:, i)$	Spans the entire i-th column of $\mathbf{X}$ (same for tensors)
$\mathbf{x}^{(k)}$	Value at the k-th iteration
`CP_ALS`	Frobenius norm PARAFAC [BK+15]
`CP_APR`	KL-Divergence PARAFAC [CK12]

Table 8.1: Table of symbols

What we have seen so far, in Chapter 7 , is the extension of CORCONDIA to large and sparse data, assuming Frobenius norm loss. This assumption postulates that the underlying data distribution is Gaussian. However, recently [CK12] showed that for sparse data that capture counts (e.g. number of messages exchanged), it is more beneficial to postulate a Poisson distribution, therefore using the KL-Divergence as a loss function. This has been more recently adopted in [HGS14] showing very promising results in medical applications. Therefore, one natural direction, which we follow in the first part of the next section, is to extend CORCONDIA for this scenario.

8.3 Proposed Methods

In exploratory data mining applications, the case is very frequently the following: we are given a piece of (usually very large) data that is of interest to a domain expert, and we are asked to identify regular and irregular patterns that are potentially useful to the expert who is providing the data. Very often, this is done in a entirely unsupervised way, since ground truth and labels are either very expensive or hard to obtain. In our context of tensor data mining, our problem, thus, is given a potentially very large and sparse tensor, and its F component decomposition, compute a quality measure for that decomposition. Subsequently, using that quality metric, we would like to identify a "good" number F of components, and throughout this process, we would like to minimize human intervention and trial-and-error fine tuning.

In order to attack the above problem, first, in Section 8.3.1 we describe how we can derive a fast and efficient measure of quality for KL-Divergence loss. Finally, in 8.3.2, we introduce AUTOTEN, our unified algorithm for automatic tensor mining with minimal user intervention, and quality characterization of the solution.

8.3.1 Quality Assessment with KL-Divergence

As we saw in the description of the Core Consistency Diagnostic with Frobenius norm loss, its computation requires solving a least squares problem. In the case of the `CP_APR` modelling, where the loss function is the KL-Divergence, the minimization problem that we need to solve is:

$$\min_{\mathbf{x}} D_{KL}(\mathbf{y}||\mathbf{W}\mathbf{x}), \;\; \mathbf{W} = \mathbf{A} \otimes \mathbf{B} \otimes \mathbf{C}. \tag{8.1}$$

Unlike the Frobenius norm case, where the solution to the problem is the Least Squares estimate, in the KL-Divergence case, the problem does not have a closed form solution. Instead, iterative solutions apply. The most prominent approach to this problem is via an optimization technique called *Majorization-Minimization* (MM) or *Iterative Majorization* [Hei95]. In a nutshell, in MM, given a function that is hard to minimize directly, we derive a "majorizing" function, which is always greater than the function to be minimized, except for a support point where it is equal; we minimize the majorizing function, and iteratively update the support point using the minimizer of that function. This procedure converges to a local minimum. For the problem of Eq. 8.1, [FI11] and subsequently [CK12], employ the following update rule for the problem, which is used iteratively until convergence to a stationary point.

$$\mathbf{x}(j)^{(k)} = \mathbf{x}(j)^{(k-1)}\left(\frac{\sum_i \mathbf{W}(i,j)\left(\frac{\mathbf{y}(j)}{\tilde{\mathbf{y}}(j)^{(k-1)}}\right)}{\sum_i \mathbf{W}(i,j)}\right) \tag{8.2}$$

where $\tilde{\mathbf{y}}^{(k-1)} = \mathbf{W}\mathbf{x}^{(k-1)}$, and k denotes the k-th iteration index.

The above solution is generic for any structure of $\mathbf{W}$. Remember, however, that $\mathbf{W}$ has very specific Kronecker structure which we should exploit. Additionally, suppose that we have a $10^4 \times 10^4 \times 10^4$ tensor; then, the large dimension of $\mathbf{W}$ will be 10^{12}. If we attempt to materialize, store, and use $\mathbf{W}$ throughout the algorithm, it will be catastrophic to the algorithm's performance. We can exploit the Kronecker structure of $\mathbf{W}$ so that we break down Eq. 8.2 into pieces, each one which can be computed efficiently, given the structure of $\mathbf{W}$. The first step is to decompose the expression of the numerator of Eq. 8.2. In particular, we equivalently write $\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} * \mathbf{z}_2$ where $\mathbf{z}_2 = \mathbf{W}^T\mathbf{z}_1$ and $\mathbf{z}_1 = \mathbf{y} \oslash \tilde{\mathbf{y}}$. Due to the Kronecker structure of $\mathbf{W}$:

$$\mathbf{z}_2 = \textsc{KronMatVec}(\{\mathbf{A}^T, \mathbf{B}^T, \mathbf{C}^T\}, \mathbf{z}_1)$$

Therefore, the update to $\mathbf{x}^{(k)}$ is efficiently calculated in the three above steps. The normalization factor of the equation is equal to: $\mathbf{s}(j) = \sum_i \mathbf{W}(i,j)$. Given the Kronecker structure of $\mathbf{W}$ however, the following holds:

Claim 8.1.:
The row sum of a Kronecker product matrix $\mathbf{A} \otimes \mathbf{B}$ can be rewritten as $\left(\sum_{i=1}^{I} \mathbf{A}(i,:)\right) \otimes \left(\sum_{j=1}^{J} \mathbf{B}(j,:)\right)$

Proof. We can rewrite the row sums $\sum_{i=1}^{I} \mathbf{A}(i,:) = \mathbf{i}_I^T\mathbf{A}$ and $\sum_{j=1}^{J} \mathbf{B}(j,:) = \mathbf{i}_J^T\mathbf{B}$ where $\mathbf{i}_I$ and $\mathbf{i}_J$ are all-ones column vectors of size I and J respectively. For the Kronecker product of the row sums and by using properties of the Kronecker product, and calling $\mathbf{A} \otimes \mathbf{B} = \mathbf{W}$ we have

$$\left(\mathbf{i}_I^T\mathbf{A}\right) \otimes \left(\mathbf{i}_J^T\mathbf{B}\right) = \left(\mathbf{i}_I \otimes \mathbf{i}_J\right)^T \left(\mathbf{A} \otimes \mathbf{B}\right) = \mathbf{i}_{IJ}^T\mathbf{W} = \sum_{i=1}^{IJ} \mathbf{W}(i,:)$$

which concludes the proof. ∎

Thus, $\mathbf{s} = \left(\sum_i \mathbf{A}(i,:)\right) \otimes \left(\sum_j \mathbf{B}(j,:)\right) \otimes \left(\sum_n \mathbf{C}(n,:)\right)$.

Putting everything together, we end up with Algorithm 8.1 which is an efficient solution to the minimization problem of Equation 8.1. As in the naive case, we also use Iterative Majorization in the efficient algorithm; we iterate updating $\mathbf{x}^{(k)}$ until we converge to a local optimum or we reach a maximum number of iterations. After computing the core tensor, we then calculate the Core Consistency Diagnostic as before, by measuring its deviation from the super-diagonal tensor.

8.3.2 AutoTen: Automated Unsupervised Tensor Mining

At this stage, we have the tools we need in order to design an automated tensor mining algorithm that minimizes human intervention and provides quality characterization of the solution. We call our proposed method AutoTen, and we view this as a step towards making tensor mining a fully automated tool, used as a black box by academic and industrial practitioners. AutoTen is a two step algorithm, where we first search through the solution space and at the second step, we automatically select a good solution

Algorithm 8.1: Efficient Majorization – Minimization for solving $\min_{\mathbf{x}} D_{KL}(\mathbf{y}||\left(\mathbf{A} \otimes \mathbf{B} \otimes \mathbf{C}\right)\mathbf{x})$

```
Input: Vector $\mathbf{y}$ and matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$.
Output: Vector $\mathbf{x}$
 1: Initialize $\mathbf{x}^{(0)}$ randomly
 2: $\tilde{\mathbf{y}} = \textsc{KronMatVec}(\{\mathbf{A}, \mathbf{B}, \mathbf{C}\}, \mathbf{x}^{(0)})$
 3: $\mathbf{s} = (\sum_i \mathbf{A}(i,:)) \otimes \left(\sum_j \mathbf{B}(j,:)\right) \otimes (\sum_n \mathbf{C}(n,:))$
 4: while convergence criterion is not met do
 5:     $\mathbf{z}_1 = \mathbf{y} \oslash \tilde{\mathbf{y}}$
 6:     $\mathbf{z}_2 = \textsc{KronMatVec}(\{\mathbf{A}^T, \mathbf{B}^T, \mathbf{C}^T\}, \mathbf{z}_1)$
 7:     $\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} * \mathbf{z}_2$
 8:     $\tilde{\mathbf{y}} = \textsc{KronMatVec}(\{\mathbf{A}, \mathbf{B}, \mathbf{C}\}, \mathbf{x}^{(k)})$
 9: end while
10: Normalize $\mathbf{x}^{(k)}$ using s
```

based on its quality and the number of components it offers. A sketch of AUTOTEN follows, and is also outlined in Algorithm 8.2 and in Figure 8.2.

Solution Search The user provides a data tensor, as well as a maximum rank that reflects the budget that she is willing to devote to AUTOTEN's search. We neither have nor require any prior knowledge whether the tensor is highly sparse, or dense, contains real values or counts, hinting whether we should use, say, `CP_ALS` postulating Frobenius norm loss, or `CP_APR` postulating KL-Divergence loss.

Fortunately, our work in this Chapter, as well as in the previous Chapter 7 has equipped us with tools for handling all of the above cases. Thus, we follow a data-driven approach, where we let the data show us whether using `CP_ALS` or `CP_APR` is capturing better structure. For a grid of values for the decomposition rank (bounded by the user provided maximum rank), we run both `CP_ALS` and `CP_APR`, and we record the quality of the result as measured by the Core Consistency diagnostic into vectors $\mathbf{c}_{Fro}$ and $\mathbf{c}_{KL}$ truncating negative values to zero.

Result Selection At this point, for both `CP_ALS` and `CP_APR` we have points in two dimensional space (F_i, c_i), reflecting the quality and the corresponding number of components. Given points (F_i, c_i) we need to find one that maximizes the quality of the decomposition, as well as finding as many hidden components in the data as possible. Intuitively, we are seeking a decomposition that discovers as many latent components as possible, without sacrificing the quality of those components. Essentially, we have a multi-objective optimization problem, where we need to maximize both c_i and F_i. However, if we, say, get the Pareto front of those points (i.e. the subset of all non-dominated points), we end up with a family of solutions without a clear guideline on how to select one. We propose to use the following, effective, two-step maximization algorithm that gives an intuitive *data-driven* solution:

- **Max c step**: Given vector **c**, run 2-means clustering on its values. This will essentially divide the vector into a set of good/high values and a set of low/bad ones. If we call m_1, m_2 the means of the two clusters, then we select the cluster index that corresponds to the maximum between m_1 and m_2.
- **Max F step**: Given the cluster of points with maximum mean, we select the point that maximizes the value of F. We call this point (F^*, c^*).

Figure 8.1 shows pictorially the output of this two-step algorithm for a set of points taken from a real dataset. Note that the choice of the above algorithm, *intuitively*, is a good compromise between the quality of the decomposition, as indicated by the CORCONDIA value, as well as the number of latent patterns that we uncover.

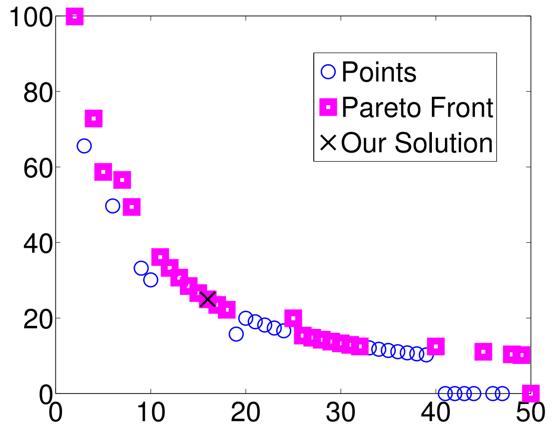


Figure 8.1: Example of choosing a good point

Another alternative is to formally define a function of c, F that we wish to maximize, and select the maximum via enumeration. Coming up with the particular function to maximize, considering the intuitive objective of maximizing the number of components that we can extract with reasonably high quality (c), is a hard problem, and we risk biasing the selection with a specific choice of a function. Nevertheless, an example such function can be $g(c, F) = logclogF$ for $c > 0$, and $g(0, F) = 0$; this function essentially measures the area of the rectangle formed by the lines connecting (F, c) with the axes (in the log-log space) and intuitively seeks to find a good compromise between maximizing F and c. This function performs closely to the proposed data-driven approach and we defer a detailed discussion and investigation to future work.

After choosing the "best" points (F^*_{Fro}, c^*_{Fro}) and (F^*_{KL}, c^*_{KL}), at the final step of AUTOTEN, we have to select between the results of `CP_ALS` and `CP_APR`. In order do so, we can use the following strategies:

1. Calculate
$$s_{Fro} = \sum_f \mathbf{c}_{Fro}(f)$$
and
$$s_{KL} = \sum_f \mathbf{c}_{KL}(f),$$
and select the method that gives the largest sum. The intuition behind this data-driven strategy is choosing the loss function that is able to discover results with higher quality on aggregate, for more potential ranks.
2. Select the results that produce the maximum value between c^*_{Fro} and c^*_{KL}. This strategy is conservative and aims for the highest quality of results, possibly to the expense of components of lesser quality that could still be acceptable for exploratory analysis.
3. Select the results that produce the maximum value between F^*_{Fro} and F^*_{KL}. Contrary to the previous strategy, this one is more aggressive, aiming for the highest number of components that can be extracted with acceptable quality.

Empirically, the last strategy seems to give better results, however they all perform very closely in synthetic data. Particular choice of strategy depends on the application needs, e.g. if quality of the components is imperative to be high, then strategy 2 should be preferred over strategy 3.

Algorithm 8.2: AUTOTEN: Automatic Unsupervised Tensor Mining

```
Input: Tensor $\underline{\mathbf{X}}$ and maximum budget for component search $F_{max}$
Output: PARAFAC decomposition $\mathbf{A}, \mathbf{B}, \mathbf{C}$ of $\underline{\mathbf{X}}$ and corresponding quality metric $c^*$.
1: for $f = 2 \cdots F_{max}$, in parallel do
2:     Run CP_ALS for $f$ components. Update $\mathbf{c}_{Fro}(f)$ using Algorithm in Chapter 7.
3:     Run CP_APR for $f$ components. Update $\mathbf{c}_{KL}(f)$ using Algorithm 8.1.
4: end for
5: Find $(F^*_{Fro}, c^*_{Fro})$ and $(F^*_{KL}, c^*_{KL})$ using the two-step maximization as described in the text.
6: Choose between CP_ALS and CP_APR using the strategy described in the text.
7: Output the chosen $c^*$ and the corresponding decomposition.
```

Discussion As we've mentioned above, the maximum number of components F_{max} is chosen by the user according to the computational budget. However, there also exist rigorous theoretical bounds on F_{max} that can help guide the choice. In particular, the main result in, as we have also seen in Chapters 2 and 6, [CO12] states that for a tensor of dimensions $I \times J \times K$, assuming $I \leq J \leq K$, the maximum number of components that can be *uniquely* identified using the PARAFAC decomposition is $F_{max} \leq \frac{(I+1)(J+1)}{16}$, which is an upper bound to the choice of the F_{max} parameter in AUTOTEN. We point out that lines 2-3 of Algorithm 8.2, i.e. all the F_{max} computations, can be run *entirely in parallel*, speeding up the computation of each individual decomposition. Finally, it is

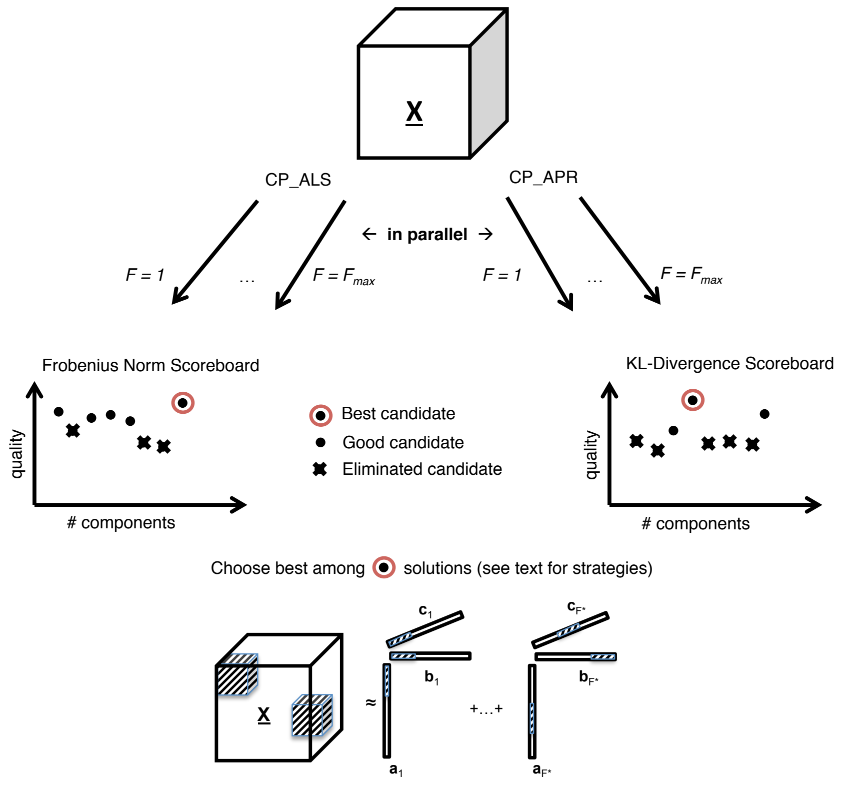


Figure 8.2: High-level pictorial description of AUTOTEN.

important to note that AUTOTEN not only seeks to find a good number of components for the decomposition, combining the best of both worlds of CP_ALS and CP_APR, but furthermore is able to provide quality assessment for the decomposition: if for a given F_{max} none of the solutions that AUTOTEN sifts through yields a satisfactory result, the user will be able to tell because of the very low (or zero in extreme cases) c^* value.

8.4 Experimental Evaluation

We implemented AUTOTEN in Matlab, using the Tensor Toolbox [BK+15], which provides efficient manipulation and storage of sparse tensors. We use the public implementation for the algorithm of [PF15], and we make our code publicly available[1]. The online version of our code contains a test case that uses the same code that we used for the following evaluation. All experiments were carried out on a workstation with 4 Intel(R) Xeon(R) E7- 8837 and 512Gb of RAM.

8.4.1 Evaluation on Synthetic Data

In this section, we empirically measure AUTOTEN's ability to uncover the true number of components hidden in a tensor. We create synthetic tensors of size $50 \times 50 \times 50$ in the

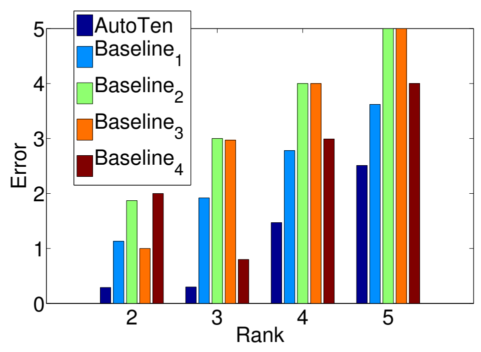
[1]Download our code at http://www.cs.cmu.edu/~epapalex/src/AutoTen.zip

two following ways that model realistic data mining scenarios where we have highly sparse data: 1) using the function `create_problem` of the Tensor Toolbox for Matlab as a standardized means of generating synthetic tensors, we generate sparse random factors with integer values, with total number of non-zeros equal to 500, 2) following the synthetic data generation methodology of [CK12], which generates poisson distributed sparse factors. We generate these for true rank F_o ranging from 2-5.

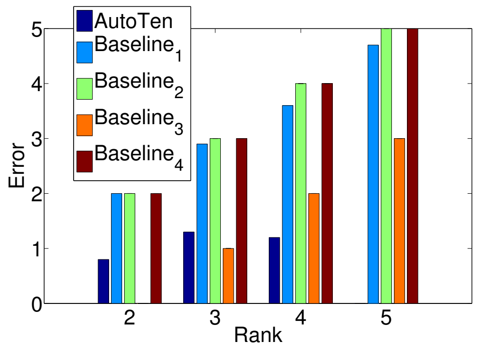
We compare AUTOTEN against four baselines:

- **Baseline 1**: A Bayesian tensor decomposition approach, as introduced very recently in [ZZC15] which automatically determines the rank.
- **Baseline 2**: This is a very simple heuristic approach where, for a grid of values for the rank, we run `CP_ALS` and record the Frobenius norm loss for each solution. If for two consecutive iterations the loss does not improve more than a small positive number ϵ (set to 10^{-6} here), we declare as output the result of the previous iteration.
- **Baseline 3**: Same as Baseline 2 with sole difference being that we use `CP_APR` and accordingly instead of the Frobenius norm reconstruction error, we measure the log-likelihood, and we stop when it stops improving more than ϵ. We expect Baseline 3 to be more effective than Baseline 2 in sparse data, due to the more delicate and effective treatment of sparse, count data by `CP_APR`.
- **Baseline 4**: A Bayesian framework based on Automatic Relevance Determination (ARD) that is adapted to the rank estimation of PARAFAC and TUCKER models [MH09]. According to [MH09] this baseline works comparably to Core Consistency in the cases the authors examined.

AUTOTEN as well as Baselines 2 & 3 require a maximum bound F_{max} on the rank; for fairness, we set $F_{max} = 2F_o$ for all three methods. In Figures 8.3(a) and 8.3(b) we show the results for both test cases. The error is measured as $|F_{est} - F_o|$ where F_{est} is the estimated number of components by each method. Due to the randomized nature of the synthetic data generation, we ran 100 iterations and we show the average results. We calculated statistical significance of our results ($p < 0.01$) using a two-sided sign test. We observe that AUTOTEN generally achieves lower error in estimating the true number of components. There is a single instance in Fig. 8.3(b) where the log likelihood criterion (Baseline 3) works slightly better than the proposed method, and this is probably because the criterion of Baseline 3 is highly in sync with the generative model of the synthetic data, however, overall we conclude that AUTOTEN largely outperforms the baselines in synthetic data that emulates realistic tensor mining applications. The problem at hand is an extremely hard one, and we are not expecting any *tractable* method to solve it perfectly. Thus, the results we obtain here are very encouraging and show that AUTOTEN is a practical solution that, as we demonstrate in the next Section, can help data mining practitioners.



(a) Sparse count data with integer factors



(b) Data generated as described in [CK12]

Figure 8.3: **AutoTen outperforms baselines**: Rank estimation error on synthetic data.

8.5 Data Mining Case Study

Section 8.5.1 takes 10 diverse real datasets shown in Table 8.2 and investigates their rank structure. In Section 8.5.2 we apply AutoTen to one of the datasets of Table 8.2 and we analyze the results, as part of an exploratory data mining study.

Name	**Description**	**Dimensions**	**Number of nonzeros**
`ENRON`	(sender, recipient, month)	$186 \times 186 \times 44$	9838
`Reality Mining` [EPL09]	(person, person, means of communication)	$88 \times 88 \times 4$	5022
`Facebook` [VMCG09]	(wall owner, poster, day)	$63891 \times 63890 \times 1847$	737778
`Taxi` [YZXS11, WZX14]	(latitude, longitude,minute)	$100 \times 100 \times 9617$	17762489
`DBLP` [PAI13]	(paper, paper, view)	$7317 \times 7317 \times 3$	274106
`Netflix`	(movie, user, date)	$17770 \times 252474 \times 88$	50244707
`Amazon co-purchase` [LK14]	(product, product, product group)	$256 \times 256 \times 5$	5726
`Amazon metadata` [LK14]	(product, customer, product group)	$10000 \times 263011 \times 5$	441301
`Yelp`	(user, business, term)	$43872 \times 11536 \times 10000$	10009860
`Airport`	(airport, airport, airline)	$9135 \times 9135 \times 19305$	58443

Table 8.2: Datasets analyzed

8.5.1 Rank Structure of Real Datasets

Since exploration of the rank structure of a dataset, using the Core Consistency diagnostic, is an integral part of AutoTen, we deem necessary to dive deeper into that process. In this case study we are analyzing the rank structure of 10 real datasets, as captured by the Core Consistency under Frobenius norm loss (using our algorithm from [PF15], as well as Core Consistency with KL-Divergence loss (introduced here). Most of the datasets we use are *publicly available*. `ENRON`[2] is a social network dataset, recording the number of emails exchanged between employees of the company for a period of time, during the company crisis. `Reality Mining` [EPL09] is a multi-view social network dataset, recording relations between MIT students (who calls whom, who messages whom, who is close to whom and so on). `Facebook` [VMCG09] is a time evolving

[2]`http://www.cs.cmu.edu/~enron/`

snapshot of Facebook, recording people posting on other peoples' Walls. `Taxi`[3] is a dataset of taxi trajectories in Beijing; we discretize latitude and longitude to a 100×100 grid. `DBLP` is a dataset recording which researcher to researcher connections, from three different viewpoints (first view is co-authorship, second view is citation, and third view records whether two authors share at least three keywords in their title or abstract of their papers). `Netflix` comes from the Netflix prize dataset and records movie ratings by users over time. `Amazon co-purchase` data records items bought together, and the category of the first of the two products. `Amazon metadata` records customers who reviewed a product, and the corresponding product category. `Yelp` contains reviews of Yelp users for various businesses (from the data challenge[4]). Finally, `Airport`[5] contains records of flights between different airports, and the operating airline.

We note that in this Chapter we are mainly investigating the structure of the above datasets. However, we also conduct data mining case studies for the majority of the aforementioned datasets in various chapters: we analyze `ENRON` and `Facebook` in Chapter 4 and `Reality Mining` and `DBLP` in Chapter 11.

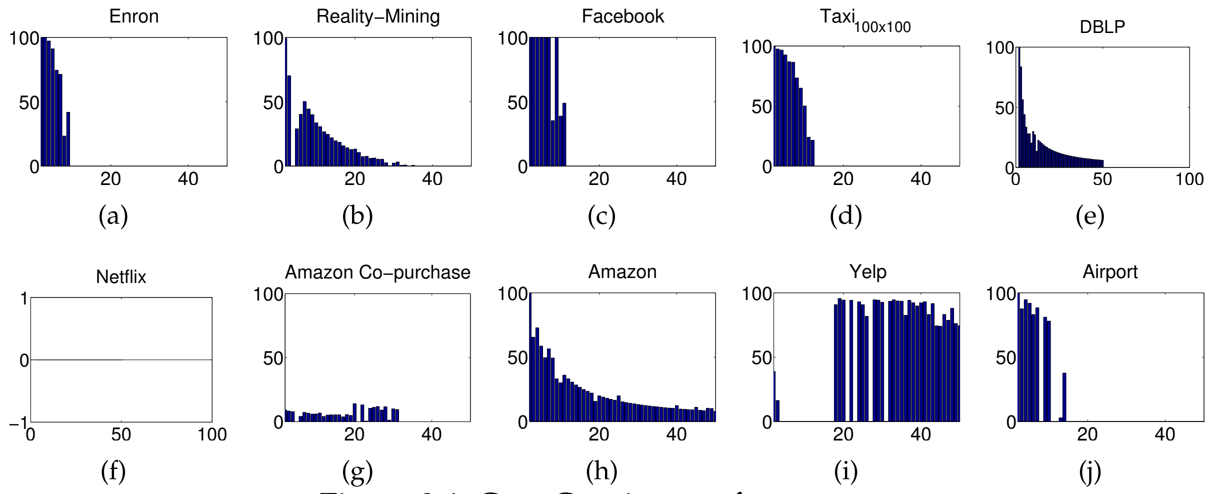


Figure 8.4: Core Consistency for `CP_ALS`

We ran our algorithms for $F = 2 \cdots 50$, and truncated negative values to zero. For KL-Divergence and datasets `Facebook`, `Netflix`, `Yelp`, and `Airport` we used smaller versions (first 500 rows for `Netflix` and `Yelp`, and first 1000 rows for `Facebook` and `Airport`), due to high memory requirements of Matlab; this means that the corresponding figures describe the rank structure of a smaller dataset, which might be different from the full one. Figure 8.4 shows the Core Consistency when using Frobenius norm as a loss, and Fig. 8.5 when using KL-Divergence. The way to interpret these figures is the following: assuming a `CP_ALS` (Fig. 8.4) or a `CP_APR` (Fig. 8.5) model, each figure shows the modelling quality of the data for a given rank. This sheds light to the rank structure

[3] `http://research.microsoft.com/apps/pubs/?id=152883`
[4] `https://www.yelp.com/dataset_challenge/dataset`
[5] `http://openflights.org/data.html`

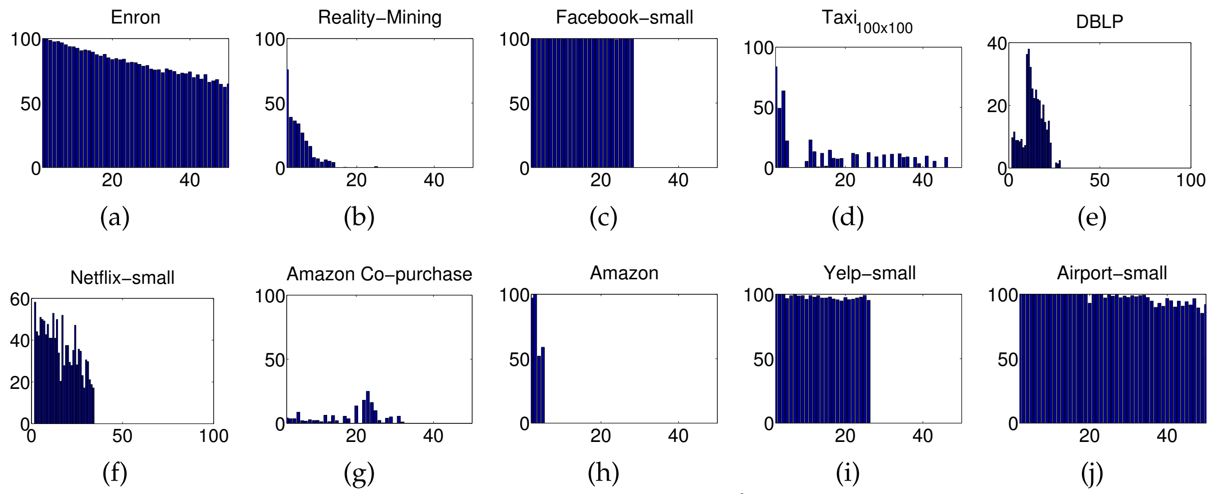


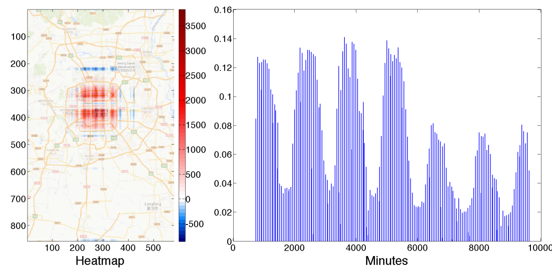
Figure 8.5: Core Consistency for `CP_APR`

of a particular dataset (although that is not to say that it provides a definitive answer about its true rank). For the given datasets, we observe a few interesting differences in structure: for instance, `ENRON` and `Taxi` in Fig. 8.4 seem to have good quality for a few components. On the other hand, `Reality Mining`, `DBLP`, and `Amazon metadata` have reasonably acceptable quality for a larger range of components, with the quality decreasing as the number gets higher. Another interesting observation is that `Yelp` seems to be modelled better using a high number of components. Figures that are all-zero merely show that no good structure was detected for up to 50 components, however, this might indicate that such datasets (e.g. `Netflix`) have an even higher number of components. Finally, contrasting Fig. 8.5 to Fig. 8.4, we observe that in many cases using the KL-Divergence is able to discover better structure than the Frobenius norm (e.g. `ENRON` and `Amazon co-purchase`).

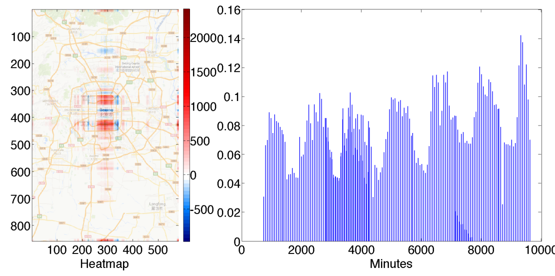
8.5.2 AutoTen in practice

We used AUTOTEN to analyze the `Taxi` dataset shown in Table 8.2.The data we have span an entire week worth of measurements, with temporal granularity of minutes. First, we tried quantizing the latitude and longitude into a 1000×1000 grid; however, AUTOTEN warned us that the decomposition was not able to detect good and coherent structure in the data, perhaps due to extreme sparsity. Subsequently, we modelled the data using a 100×100 grid and AUTOTEN was able to detect good structure. In particular, AUTOTEN output 8 rank-one components, choosing Frobenius norm as a loss function.

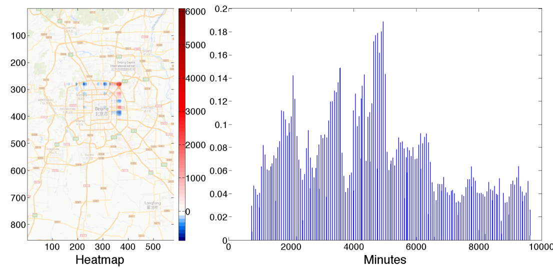
In Figure 8.6 we show 4 representative components of the decomposition. In each sub-figure, we overlay the map of Beijing with the coordinates that appear to have high activity in the particular component; every sub-figure also shows the temporal profile of the component. The first two components (Fig. 8.6(a), (b)) spatially refer to a similar area, roughly corresponding to the tourist and business center in the central rings of



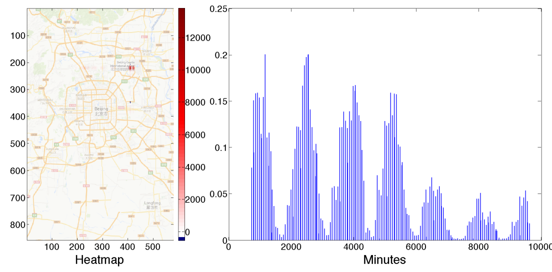
(a) **Tourist & Business Center**: High activity during weekdays, low over the weekend



(b) **Downtown**: Consistent activity throughout the week



(c) **Olympic Center**: Activity peak during the week



(d) **Airport**: High activity during weekdays, low over the weekend

Figure 8.6: Latent components of the `Taxi` dataset, as extracted using AUTOTEN.

the city. The difference is that Fig. 8.6(a) shows high activity during the weekdays and declining activity over the weekend (indicated by five peaks of equal height, followed by two smaller peaks), whereas Fig. 8.6(b) shows a slightly inverted temporal profile, where the activity peaks over the weekend; we conclude that Fig. 8.6(a) most likely captures business traffic that peaks during the week, whereas Fig. 8.6(b) captures tourist and leisure traffic that peaks over the weekend. The third component (Fig. 8.6(c)) is highly active around the Olympic Center and Convention Center area, with peaking activity in the middle of the week. Finally, the last component (Fig. 8.6(d)) shows high activity only outside of Beijing's international airport, where taxis gather to pick-up customers; on the temporal side, we see daily peaks of activity, with the collective activity dropping during the weekend, when there is significantly less traffic of people coming to the city for business. By being able to analyze such trajectory data into highly interpretable results, we can help policymakers to better understand the traffic patterns of taxis in big cities, estimate high and low demand areas and times and optimize city planning in that respect. There has been very recent work [WZX14] towards the same direction, and we view our results as complementary.

8.6 Related Work

We have already covered model order selection in Chapter 2, however, for the reader's convenience, we outline here the most relevant references. As we have mentioned throughout the text, CORCONDIA [BK03] is using properties of the PARAFAC decomposition in order to hint towards the right number of components. In [PF15], the authors introduce a scalable algorithm for CORCONDIA (under the Frobenius norm). Moving away from the PARAFAC decompostion, Kiers and Kinderen [KK03] introduce a method for choosing the number of components for TUCKER. There has been recent work using Minimum Description Length (MDL): In [APG+14] the authors use MDL in the context of community detection in time-evolving social network tensors, whereas in [MM15], Metzler and Miettinen use MDL to score the quality of components for a binary tensor factorization. Finally, there have also been recent advances using Bayesian methods in order to automatically decide the number of components: in particular [ZZC15] does so for the PARAFAC decomposition, and [MH09] (based on Automatic Relevance Determination) does so for both PARAFAC and TUCKER models.

8.7 Conclusions

In this Chapter, we work towards an automatic, unsupervised tensor mining algorithm that minimizes user intervention. We encourage *reproducibility* by making our code publicly available at `http://www.cs.cmu.edu/~epapalex/src/AutoTen.zip`. Our main contributions are:

- **Technology Transfer** This is the first work to apply ideas such as the Core Consistency Diagnostic [BK03] in data mining, aiming to popularize it within the community.
- **Algorithms** We propose AUTOTEN, a novel automatic, parallel, and unsupervised tensor mining algorithm, which can provide quality characterization of the solution. We extend the Core Consistency Diagnostic of [BK03] for KL-Divergence loss and provide a novel and efficient algorithm.
- **Evaluation & Discovery** We evaluate our methods in synthetic data, showing their superiority compared to the baselines, as well as a wide variety of real datasets. Finally, we apply AUTOTEN to two real datasets discovering meaningful patterns (Section 8.5.2 and supplementary material).

Part III

Applications: Neurosemantics

Chapter based on material published in [PMS+14, PMS+16].

Chapter 9

Coupling Brain Measurements with Semantic Information

Unsupervised discovery of semantically coherent brain regions.

Consider the following experimental setting, where multiple human subjects are shown a set of concrete English nouns (e.g. "dog", "tomato" etc), and we measure each person's brain activity using functional Magnetic Resonance Imaging (fMRI). Here we identify regions of the brain that are activated for a semantically coherent set of stimuli using Coupled Matrix-Tensor Factorization and TURBO-SMT (Chapter 5). Our analysis produces results that agree with Neuroscience and does so without supervision.

9.1 Introduction

How is knowledge represented in the human brain? Which regions have high activity, when a person sees a concept such as "animal" or "food"? In order to answer the above questions, we consider the following experiment: multiple human subjects are shown a set of concrete English nouns (e.g. "dog", "tomato" etc), and we measure each person's brain activity using functional Magnetic Resonance Imaging (fMRI). In this experiment, human subjects and semantic stimuli (i.e., the nouns) different aspects of the same underlying phenomenon: the mechanisms that the brain uses to process language.

We seek to identify coherent regions of the brain that are activated for a semantically coherent set of stimuli. To that end we *couple* the above fMRI measurements with semantic features for the same set of nouns, which offer another aspect of the data and provide useful information to the analysis which might be missing from the fMRI data.

In the following lines we describe the data we use and the way we formulate the problem of identifying semantically coherent brain regions (Sec. 9.2), we present our discoveries (Sec. 9.3) and conclude outlining the importance of our results.

9.2 Data Description & Problem Formulation

Our data consists of two parts, the fMRI brain scans and the semantic information for the stimuli of the experiment.

9.2.1 fMRI Brain Scans

We record the brain activity of 9 human subjects when shown each of 60 concrete nouns (5 in each of 12 categories, e.g. dog, hand, house, door, shirt, dresser, butterfly, knife, telephone, saw, lettuce, train). The human subjects were balanced in terms of gender and demographics, and were all right-handed. We use fMRI for the measurements; fMRI measures slow changes in blood oxygenation levels, reflecting localized changes in brain activity. The data produced consists of $3 \times 3 \times 6$mm voxels (3D pixels) corresponding to fixed spatial locations across participants. Recorded fMRI values are the mean activity over 4 contiguous seconds, averaged over multiple presentations of each stimulus word (each word is presented 6 times as a stimulus). Further acquisition and preprocessing details are given in [MSC+08]. This dataset is publicly available[1].

For each human subject, noun pair we take the fMRI image and we "flatten" it to a long vector that contains the intensity values for all the voxels. Using this representation, we create a (noun, voxel, human subject) tensor $\underline{\mathbf{X}}$ of dimensions $60 \times 77775 \times 9$ with over 11 million non-zeros, that contains the measurements for all human subjects and all stimuli.

9.2.2 Semantic Information

So far, we have represented the experiment as a tensor, however, we are still lacking a notion of semantics. By taking a low-rank, sparse PARAFAC decomposition of $\underline{\mathbf{X}}$ we will obtain subsets of nouns that activate similar parts of the brain for similar subsets of human subjects. However, we further need a means of understanding the semantic coherence of each one of those latent groups.

To that end, we use a separate piece of data that contains crowd-sourced numerical responses to 218 questions (such as "can you buy it?", "does it fly"?, and"is it smaller than a golfball?"), for each of the 60 nouns. This dataset has been used before in works such as [MTM12], [PPHM09]. We stress that the set of people who gave the answers to the above questions is different from the 9 human subjects of our brain scans. However, using this dataset is beneficial for our analysis in the following ways:

- The questions are semantic features that constitute a *human-readable description* of a noun or a group of nouns. Thus, if we can correlate the brain scan data with the

[1]http://www.cs.cmu.edu/afs/cs/project/theo-73/www/science2008/data.html

questions, we can have an annotation of the latent cluster with a set of semantic features that describes the semantic concept of the cluster.
- The fMRI data are usually noisy and if we can use any additional, structured, side information (such as the questions), this will enable more robust discovery of meaningful structure in the fMRI data.

We must note here that this particular choice of semantic features for the nouns is not the only one. Another example of such side information could be corpus statistics for the nouns, or ontology/knowledge base information.

The questions form a (noun, question) matrix $\mathbf{Y}$ of dimensions 60×218.

9.2.3 Problem Formulation

As we describe in the lines above, we have represented the fMRI brain scans as a (noun, voxel, human subject) tensor $\underline{\mathbf{X}}$, and the questions as a (noun, questions) matrix $\mathbf{Y}$. A crucial observation is that $\underline{\mathbf{X}}$ and $\mathbf{Y}$ share the "nouns" mode (since the nouns in both datasets have one-to-one correspondence), hence they are *coupled* in the noun dimension. As we saw in Chapter 2, there exists a class of models by the name of Coupled Matrix-Tensor Factorizations (CMTF) that can handle such coupled data.

In our case, we seek to identify co-clusters of nouns, brain voxels, human subjects, and questions, thus a CMTF model with a PARAFAC model for the tensor $\underline{\mathbf{X}}$ is the most appropriate. Figure 9.1 demonstrates our analysis.

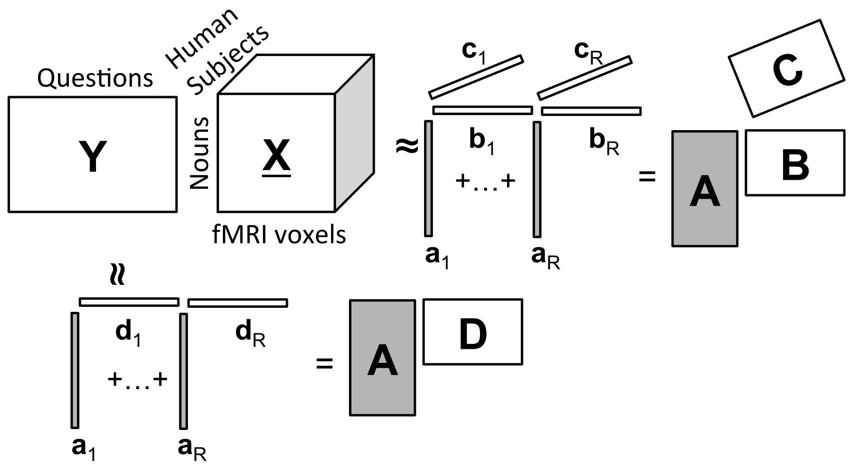


Figure 9.1: **Neurosemantics as Coupled Matrix-Tensor Factorization**: We couple the fMRI tensor $\underline{\mathbf{X}}$ and the questions matrix $\mathbf{Y}$, and we analyze the two datasets using Coupled Matrix Tensor Factorization. Each set of latent variables $\{\mathbf{a}_r, \mathbf{b}_r, \mathbf{c}_r, \mathbf{d}_r\}$ gives a subset of nouns, voxels, human subjects, and questions, identifying semantically coherent regions of the brain that responds similarly to a set of nouns, and for a particular set of people.

9.3 Discoveries

We use TURBO-SMT for analyzing $\underline{\mathbf{X}}$ and $\mathbf{Y}$, choosing $r = 5$ and $s_I = 3,\ s_J = 86,\ s_K = 1$ for the tensor and s_I for the questions dimension of the matrix [2]. In the following lines we present our discoveries.

9.3.1 Simultaneous Clustering of Words, Questions and Regions of the Brain

One of the strengths of CMTF is its expressiveness in terms of simultaneously soft-clustering all involved entities of the problem. By taking a low rank decomposition of coupled data , we are able to find groups that jointly express words, questions and brain voxels (we can also derive groups of human subjects; however, it is an active research subject in neuroscience, whether brain-scans should differ significantly between people, and is out of the scope of the present work).

In Figure 9.2, we display four such groups of brain regions that are activated given a stimulus of a group of similar words; we also display the most prominent words, along with groups of similar questions that were highly correlated with the words of each group. Moreover, we were able to successfully identify high activation of the *premotor cortex* in Group 3, which is associated with concepts that have conventional manual uses (such as holding, or picking up).

The above results is very important and encouraging because it is entirely *unsupervised* and it agrees with what Neuroscience knows about the premotor cortex. This gives confidence to our model and its potential to be used in data from more complex experiments, where it can directly help and inform Neuroscientific research.

9.3.2 Predicting Brain Activity from Questions

In addition to soft-clustering, the low rank joint decomposition of the data offers another significant result. This low dimensional embedding of the data into a common semantic space, enables the prediction of, say, the brain activity of a subject, for a given word, given the corresponding vector of question answers for that word. In particular, we use this linear transformation in order to map the question answer vector to the latent semantic space and then expanding it to the brain voxel space, we obtain a fairly good prediction of the brain activity. This linear transformation is reminiscent of the mapping done in Latent Semantic Analysis (LSA) [DDF+90], where the authors compute a low rank embedding of terms and documents, and given a query consisting of terms, they project it to the latent dimension, taking advantage of the compaction offered by the low rank approximation. In the case of LSA, the embeddings used are orthonormal (since they are computed using SVD), and thus the "mapping" is a proper projection to the subspace. In our case, we do not restrict the embeddings to be orthonormal, and thus we perform a linear transformation.

[2]We may use imbalanced sampling factors, especially when the data is far from being 'rectangular'.

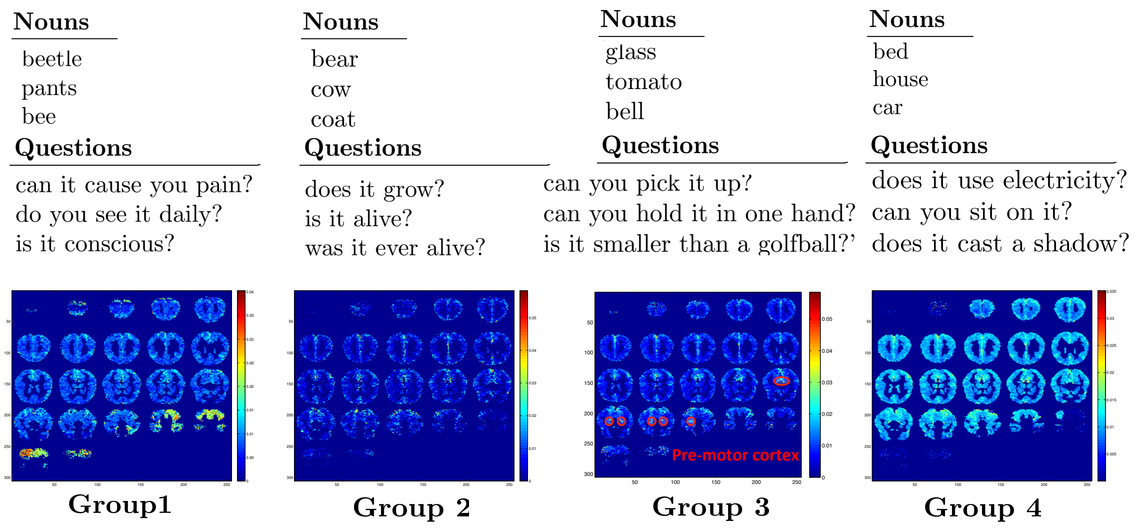


Figure 9.2: **Unsupervised discovery of semantically coherent brain regions**: TURBO-SMT finds meaningful groups of words, questions, and brain regions that are (both negatively and positively) correlated, as obtained using TURBO-SMT. For instance, Group 3 refers to small items that can be held in one hand, such as a tomato or a glass, and the activation pattern is very different from the one of Group 1, which mostly refers to insects, such as bee or beetle. Additionally, Group 3 shows high activation in the *premotor cortex* which is associated with the concepts of that group.

To evaluate the accuracy of these predictions of brain activity, we follow a *leave-two-out* scheme, where we remove two words entirely from the brain tensor and the question matrix; we carry out the joint decomposition, in some very low dimension, for the remaining set of words and we obtain the usual set of matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$. Due to the randomized nature of TURBO-SMT, we did 100 repetitions of the procedure described below.

Let $\mathbf{q}_i$ be the question vector for some word i, and $\mathbf{v}_i$ be the brain activity of one human subject, pertaining to the same word. By left-multiplying $\mathbf{q}_i$ with $\mathbf{D}^T$, we map $\mathbf{q}_i$ to the latent space of the decomposition; then, by left-multiplying the result with $\mathbf{B}$, we map the result to the brain voxel space. Thus, our estimated (predicted) brain activity is obtained as $\hat{\mathbf{v}}_i = \mathbf{B}\mathbf{D}^T\mathbf{q}_i$

Given the predicted brain activities $\hat{\mathbf{v}}_\mathbf{1}$ and $\hat{\mathbf{v}}_\mathbf{2}$ for the two left out words, and the two actual brain images $\mathbf{v}_\mathbf{1}$ and $\mathbf{v}_\mathbf{2}$ which were withheld from the training data, the *leave-two-out* scheme measures prediction accuracy by the ability to choose which of the observed brain images corresponds to which of the two words. After mean-centering the vectors, this classification decision is made according to the following rule:

$$\|\mathbf{v}_1 - \hat{\mathbf{v}}_1\|_2 + \|\mathbf{v}_2 - \hat{\mathbf{v}}_2\|_2 < \|\mathbf{v}_1 - \hat{\mathbf{v}}_2\|_2 + \|\mathbf{v}_2 - \hat{\mathbf{v}}_1\|_2$$

Although our approach is not designed to make predictions, the results are very encouraging: Using only F=2 components, for the noun pair *closet/watch* we obtained mean accuracy of 0.82 for 5 out of the 9 human subjects. Similarly, for the pair *knife/beetle*, we achieved accuracy of about 0.8 for a somewhat different group of 5 subjects. For the rest of the human subjects, the accuracy is considerably lower, however, it may be the case that brain activity predictability varies between subjects, a fact that requires further investigation.

9.4 Related Work

As we have outlined in Chapter 3, there has been substantial related work, which utilizes tensors for analyzing brain data. In [AABB$^+$07] Acar et al. apply the PARAFAC decomposition on EEG measurements in order to detect epileptic seizures. Morup et al. in [MHA$^+$08] introduce a variation of the PARAFAC decomposition that is able to handle temporal shifts across brain measurements. In [CZPA09], the authors describe how one can use non-negative tensor factorization for source separation and analysis of brain signals, and finally, in [DGCW13] et al. introduce a constrained PARAFAC model in order to identify the connectivity between different brain regions.

9.5 Conclusions

In this Chapter we identify *semantically coherent brain regions*, using fMRI brain scans and semantic features for the stimuli used during the experiment. Our main contributions are:

- **Problem formulation**: To the best of our knowledge, this is the first study to formulate the problem as an instance of Coupled Matrix-Tensor Factorization.
- **Unsupervised discovery**: Our findings are entirely unsupervised and agree with Neuroscience. This gives confidence that our formulation can be extended for more complex tasks where Neuroscience does not have the full picture yet, and thus help and inform Neuroscientific research

Chapter based on material published in [EFS+] and [PFS+14b].

Chapter 10

Good-Enough Brain Model: Challenges, Algorithms and Discoveries in Multi-Subject Experiments

Inspired by Control Theory & System Identification to estimate the functional connectivity of the human brain.

Given a simple noun such as "apple", and a question such as "is it edible?", what are the interactions between (groups of) neurons, also known as *functional connectivity*? In this Chapter we introduce a simple, novel *good-enough* brain model (GEBM) which effectively models the dynamics of the neuron interactions and captures the functional connectivity. We evaluate GEBM using real brain data and our results agree with Neuroscience.

10.1 Introduction

Consider human subject 'Alice', who is reading a typed noun (e.g. apple) and has to answer a simple yes/no question (e.g. *can you eat it?*). How do different parts of Alice's brain communicate during this task? This type of communication is formally defined as *functional connectivity* of the human brain and is an active field of research in Neuroscience. Coming up with a comprehensive model that captures the dynamics of the functional connectivity is of paramount importance. Achieving that, will effectively provide a better understanding of how the human brain works and can have a great impact both on machine and human learning. In this work, we tackle the above problem in a scenario where multiple human subjects are shown simple nouns and answer a set of questions for each noun, and we record their brain activity using magentoencephalography (MEG).

Although this might seem like a problem that is of sole interest to the field of Neuroscience, in fact, can also benefit from a Big Data approach. For each human subject, what we essentially have is a set of MEG sensors recording a time series of the magnetic activity of their brain, and our ultimate goal is to infer a (hidden) underlying network between different regions of their brain. Effectively, we are dealing with a problem of the broad family of *Network Discovery*, which is an active field of Big Data research; for instance, in [VVA+06] the authors induce a communication network between VoIP users through time-series analysis, while in [SKB12] the authors infer a friendship network between Twitter users by using various signals of user behavior.

Besides the nature of the problem itself, the data involved in our problem call for Big Data techniques: For each human subject an experiment involves recording of their brain activity for all combinations of nouns and questions, which results in a big number of data points that need to be processed, summarized and analyzed; as the task becomes more complicated (e.g. the subject has to read an entire sentence, or even a book instead of a single typed noun, or even look at a picture), the data volume and complexity explode. In addition to that, different measurement techniques have complementary strengths (e.g. MEG has fine time granularity but poor spatial resolution, and conversely fMRI has very high, 1mm spatial resolution but poor temporal resolution) and we would like to collect and exploit all potential benefits from all techniques, a factor that significantly increases the size of the problem; for instance, even for a small number of human subjects, all measurements span hundreds of Gigabytes. Thus, it is important to view this problem through a Big Data analytics lens, and this article constitutes a step towards this direction.

Our approach: Discovering the multi-billion connections among the tens of billions [WH88, ACG+09] of neurons would be the holy grail, and clearly outside the current scientific and technological capabilities. How close can we approach this ideal? We propose to use a *good-enough* approach, and try to explain as much as we can, by assuming a small, manageable count of neuron-regions and their interconnections, and trying to guess the connectivity from the available MEG data. In more detail, we propose to formulate the problem as 'system identification' from control theory, and we develop novel algorithms to find *sparse* solutions.

We show that our *good-enough* approach is a very good first step, leading to a tractable, yet effective model (GEBM), that can answer the above questions. Figure 10.1 gives the high-level overview: at the bottom-right, the blue, dashed-line time sequences correspond to measured brain activity; the red lines correspond to the guess of our GEBM model. Notice the qualitative goodness of fit. At the top-right, the arrows indicate interaction between brain regions that our analysis learned, with the weight being the strength of interaction. Thus we see that the vision cortex ('occipital lobe') is well connected to the language-processing part ('temporal lobe'), which agrees with neuroscience, since all our

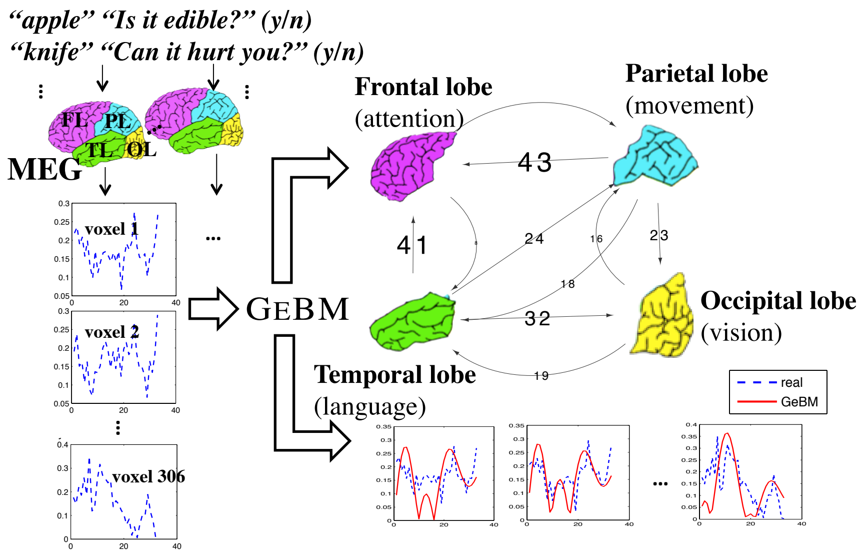


Figure 10.1: Big picture: our GEBM estimates the hidden functional connectivity (top right, weighted arrows indicating number of inferred connections), when given multiple human subjects (left) that respond to yes/no questions (e.g., *edible?*) for typed words (e.g., *apple*). Bottom right: GEBM also produces brain activity (in solid-red), that matches reality (in dashed-blue).

experiments involved typed words.

Our **contributions** are as follows:

- **Novel analytical model & Algorithm**: We propose the GEBM model (see Section 10.3, and Eq (10.2)-(10.3)). We also introduce SPARSE-SYSID, a novel sparse system-identification algorithm (see Section 10.3).
- **Effectiveness**: Our model can explain psychological phenomena, such as habituation and priming (see Section 10.5.4); it also gives results that agree with experts' intuition (see Section 10.5.1)
- **Multi-subject analysis**: Our SPARSE-SYSID, applied on 9 human subjects (Section 10.5.2), showed that (a) 8 of them had very consistent brain-connectivity patterns while (b) the outlier was due to exogenous factors (excessive road-traffic noise during his experiment).
- **Cross-disciplinary connections**: Our GEBM highlights connections between multiple, mostly disparate areas: 1) Neuroscience, 2) Control Theory & System Identification, and 3) Psychology. Additionally, we provide insights on the relation of GEBM to Recurrent Neural Networks, a field that is gaining increasing popularity among Big Data techniques especially with the rise of Deep Learning, pointing out ways that both can benefit from each other.

Reproducibility: Our implementation is publicly available [1]. Due to privacy reasons,

[1] http://www.cs.cmu.edu/~epapalex/code.html

we are not able to release the MEG data, however, in the online version of the code we include the synthetic benchmarks, as well as the simulation of psychological phenomena using GEBM.

The present chapter is an extension upon our work that appeared in the ACM KDD 2014 conference [PFS+14a]. In addition to [PFS+14a], in this manuscript, we provide further cross-disciplinary connections of our work to other fields (and more specifically to Recurrent Neural Networks), provide intuitive explanations behind the key concepts introduced in the original paper, as well as emphasize the practical applications of our work.

10.2 Problem Definition

As mentioned earlier, our goal is to infer the brain connectivity, given measurements of brain activity on multiple *yes/no tasks*, of multiple subjects. We define as **yes/no task** the experiment where the subject is given a yes/no question (like, *'is it edible?'*, *'is it alive?'*), and a typed English word (like, *apple*, *chair*), and has to decide the answer.

Throughout the entire process, we attach m sensors that record brain activity of a human subject. Here we are using Magnetoencephalography (MEG) data, although our GEBM model could be applied to other types of measurement (fMRI, etc). In Section 10.5 we provide a more formal definition of the measurement technique.

Thus, in a given experiment, at every time-tick t we have m measurements, which we arrange in an $m \times 1$ vector $\mathbf{y}(t)$. Additionally, we represent the stimulus (e.g. *apple*) and the task (e.g. *is it edible?*) in a time-dependent vector $\mathbf{s}(t)$, by using feature representation of the stimuli; a detailed description of how the stimulus vector is formed can be found in Section 10.5. For the rest of the chapter, we shall use interchangeably the terms *sensor*, *voxel* and neuron-region.

First and foremost, we are interested in understanding how the brain works, given a single subject. Informally, we have:

Informal Problem 10.1.: - **Given**: The input stimulus; and a sequence of $m \times T$ brain activity measurements for the m voxels, for all timeticks $t = 1 \cdots T$

- **Estimate**: the functional connectivity of the brain, i.e. the strength and direction of interaction, between pairs of the m voxels, such that
  1. we understand how the brain-regions collaborate, and
  2. we can effectively simulate brain activity.

After solving the above problem, we are also interested in doing cross-subject analysis, to find commonalities (and deviations) in a group of several human subjects.

For the particular experimental setting, prior work [SPP+12] has only considered transformations from the space of noun features to the voxel space and vice versa, as well as word-concept specific prediction based on estimating the covariance between the voxels [FFDM12].

Next we formalize the problems, we show some straightforward (but preliminary) solutions, and finally we give the proposed model GEBM, and the estimation algorithm.

10.3 Problem Formulation and Proposed Method

There are two over-arching assumptions in our design:

- Linearity: Linear models, however simplifying, are a good "first order approximation" of the functional connectivity we seek to capture.
- Stationarity: The connectivity of the brain does not change, at least for the time-scales of our experiments.

The above assumptions might sound very simplifying, however, this work is a first step towards this direction, and our linear, time-invariant model proves to be "*good-enough*" for the task at hand Non-linear/sigmoid models are a natural direction for future work; and so is the study of neuroplasticity, where the connectivity changes.

We must note that however simple GEBM is, there are simpler approaches which seem more natural at first, which do not perform well. Such an approach, which we call MODEL$_0$, is presented in the next subsection, before the introduction of GEBM.

10.3.1 First (unsuccessful) approach: Model$_0$

Since our Informal Problem Definition does not strictly define the brain regions whose connectivity we are seeking to identify, a natural first step is to assume that each MEG voxel (e.g. region measured by an MEG sensor) is such a brain region.

Given the linearity and static-connectivity assumptions above, we may postulate that the $m \times 1$ brain activity vector $\mathbf{y}(t+1)$ depends *linearly*, on the activities of the previous time-tick $\mathbf{y}(t)$, and, of course, the input stimulus, that is, the $s \times 1$ vector $\mathbf{s}(t)$.

Formally, in the absence of input stimulus, we expect that

$$\mathbf{y}(t+1) = \mathbf{A}\mathbf{y}(t).$$

where $\mathbf{A}$ is the $m \times m$ connectivity matrix of the m brain regions. Including the (linear) influence of the input stimulus $\mathbf{s}(t)$, we reach the MODEL$_0$:

$$\mathbf{y}(t+1) \quad = \quad \mathbf{A}_{[m\times m]} \times \mathbf{y}(t) \ + \ \mathbf{B}_{[m\times s]} \times \mathbf{s}(t) \tag{10.1}$$

The $\mathbf{B}_{[m\times s]}$ matrix shows how the s input signals affect the m brain-regions.

In [PFS+14a] we show how we can solve for this model using Least Squares (LS) and Canonical Correlation Analysis (CCA) [SGL07]. However intuitive, the formulation of MODEL$_0$ turns out to be rather ineffective in capturing the temporal dynamics of the recorded brain activity. As an example of its failure to model brain activity successfully,

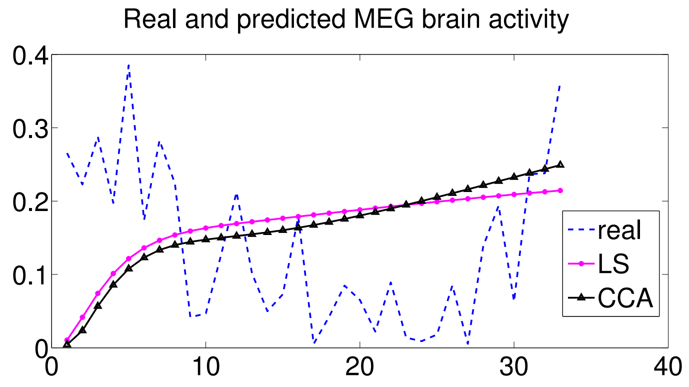


Figure 10.2: **Model$_0$ fails**: True brain activity (dotted blue) and the model estimate (pink, and black, resp., for the least squares, and for the CCA variation).

Fig. 10.2 shows the real brain activity and predicted activity for a particular voxel. The solutions fail to match the trends and oscillations.

The conclusion of this subsection is that we need a more sophisticated yet parsimonious model, which leads us to GeBM, next.

10.3.2 Proposed approach: GeBM

Before we introduce our proposed model, we should introduce our notation, which is succinctly shown in Table 10.1.

Symbol	Definition
n	number of hidden neuron-regions
m	number of MEG sensors/voxels we observe (306)
s	number of input signals (40 questions)
T	time-ticks of each experiment (340 ticks, of 5msec each)
$\mathbf{x}(t)$	vector of neuron activities at time t
$\mathbf{y}(t)$	vector of voxel activities at time t
$\mathbf{s}(t)$	vector of input-sensor activities at time t
$\mathbf{A}_{[n\times n]}$	connectivity matrix between neurons (or neuron regions)
$\mathbf{C}_{[m\times n]}$	summarization matrix (neurons to voxels)
$\mathbf{B}_{[n\times s]}$	perception matrix (sensors to neurons)
$\mathbf{A}_v$	connectivity matrix between voxels
Real	real part of a complex number
Imag	imaginary part of a complex number
$\mathbf{A}^\dagger$	Moore-Penrose Pseudoinverse of $\mathbf{A}$

Table 10.1: Table of symbols

Formulating the problem as Model$_0$ is not able to meet the requirements for our desired solution. However, we have not exhausted the space of possible formulations that live within our set of simplifying assumptions. In this section, we describe GEBM, our proposed approach which, under the assumptions that we have already made in Section 10.2, is able to meet our requirements remarkably well.

In order to come up with a more accurate model, it is useful to look more carefully at the actual system that we are attempting to model. In particular, the brain activity vector $\mathbf{y}$ that we observe is simply the collection of values recorded by the m sensors, placed on a person's scalp. In MODEL$_0$, we attempt to model the dynamics of the sensor measurements directly. However, by doing so, we are directing our attention to an *observable proxy* of the process that we are trying to estimate (i.e. the functional connectivity). Instead, it is more beneficial to model the direct outcome of that process. Ideally, we would like to capture the dynamics of the internal state of the person's brain, which, in turn, cause the effect that we are measuring with our MEG sensors.

Let us assume that there are n *hidden* (hyper-)regions of the brain, which interact with each other, causing the activity that we observe in $\mathbf{y}$. We denote the vector of the hidden brain activity as $\mathbf{x}$ of size $n \times 1$. Then, by using the same idea as in MODEL$_0$, we may formulate the temporal evolution of the hidden brain activity as:

$$\mathbf{x}(t+1) = \mathbf{A}_{[n\times n]} \times \mathbf{x}(t) \; + \; \mathbf{B}_{[n\times s]} \times \mathbf{s}(t)$$

A subtle issue that we have yet to address is the fact that $\mathbf{x}$ is *not observed* and we have no means of measuring it. We propose to resolve this issue by modelling the measurement procedure itself, i.e. model the transformation of a hidden brain activity vector to its observed counterpart. We assume that this transformation is linear, thus we are able to write

$$\mathbf{y}(t) = \mathbf{C}_{[m\times n]}\mathbf{x}(t)$$

Putting everything together, we end up with the following set of equations, which constitute our proposed model GEBM:

$$\mathbf{x}(t+1) = \mathbf{A}_{[n\times n]} \times \mathbf{x}(t) \; + \; \mathbf{B}_{[n\times s]} \times \mathbf{s}(t) \tag{10.2}$$
$$\mathbf{y}(t) = \mathbf{C}_{[m\times n]} \times \mathbf{x}(t) \tag{10.3}$$

The key concepts behind GEBM are:

- **(Latent) Connectivity Matrix**: We assume that there are n regions, each containing 1 or more neurons, and they are connected with an $n \times n$ adjacency matrix $\mathbf{A}_{[n\times n]}$. We only observe m voxels, each containing multiple regions, and we record the activity (eg., magnetic activity) in each of them; this is the total activity in the constituent regions
- **Measurement Matrix**: Matrix $\mathbf{C}_{[m\times n]}$ is an $m \times n$ matrix, with $c_{i,j}$ =1 if voxel i contains region j
- **Perception Matrix**: Matrix $\mathbf{B}_{[n\times s]}$ shows the influence of each sensor to each neuron-region. The input is denoted as s, with s input signals

- **Sparsity**: We require that our model's matrices are *sparse*; only few sensors measure a specific brain region. Additionally, the interactions between regions should not form a complete graph, and finally, the perception matrix should map only few activated sensors to neuron regions at every given time.

An interesting aspect of our proposed model GeBM is the fact that if we ignore the notion of the summarization, i.e. matrix $\mathbf{C} = \mathbf{I}$, then our model is reduced to the simple model Model$_0$. In other words, GeBM contains Model$_0$ as a special case. This observation demonstrates the importance of hidden states in GeBM.

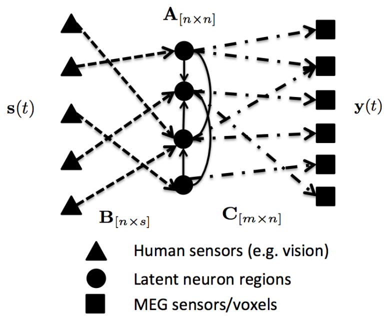

$\mathbf{A}_{[n\times n]}$
$\mathbf{s}(t)$
$\mathbf{y}(t)$
$\mathbf{B}_{[n\times s]}$
$\mathbf{C}_{[m\times n]}$
Human sensors (e.g. vision)
Latent neuron regions
MEG sensors/voxels

Figure 10.3: Sketch of GeBM

A pictorial representation of GeBM is shown in Fig. 10.3. Starting from left to right, we have the triangle shaped nodes; these nodes represent the sensors of the human subject, which capture the stimulus $\mathbf{s}(t)$. For instance, these sensors could correspond at a high level to different visual sensors that map the stimulus to the internal, hidden neuron regions; the mapping between the human sensor nodes to the latent brain regions is encoded in matrix $\mathbf{B}$. The hidden neuron regions are denoted by black circles, and the connections between them comprise matrix $\mathbf{A}$, which is effectively the functional connectivity of these latent regions. Finally, the latent region activity $\mathbf{x}(t)$ is measured by the MEG sensors, which are denoted by black squares in Fig. 10.3; this measurement procedure is encoded in matrix $\mathbf{C}$.

10.3.3 Algorithm

Our solution is inspired by control theory, and more specifically by a sub-field of control theory, called *system identification*. We refer the interested reader to the appendix of our KDD paper [PFS$^+$14a], and references therein, for an overview of traditional system identification. However, the matrices we obtain through this process are usually dense, counter to GeBM's specifications. We, thus, need to refine the solution until we obtain the desired level of sparsity. In the next few lines, we show why this sparsification has to be done carefully, and we present our approach.

Crucial to GeBM's behavior is the spectrum of its matrices; in other words, any operation that we apply on any of GeBM's matrices needs to preserve the eigevnalue profile (for matrix $\mathbf{A}$) or the singular values (for matrices $\mathbf{B}, \mathbf{C}$). Alterations thereof may lead GeBM to instabilities. From a control theoretic and stability perspective, we are mostly interested in the eigenvalues of $\mathbf{A}$, since they drive the behavior of the system. Thus, in our experiments, we heavily rely on assessing how well we estimate these eigenvalues.

As a reminder to the reader, here we provide a short explanation why eigenvalues are crucial. Say we focus on one of the eigenvalues λ of our matrix $\mathbf{A}$. If λ is real and positive, then the response of the system (associated to λ) will increase exponentially. Conversely, if λ is real and negative, the respective response will decay exponentially. Finally, if λ is complex, then the response of the system will be some type of oscillation, whose frequency and trend (decaying, increasing or constant) depends on the actual values of the real and the imaginary parts. The response of the system is, thus, a mixture of the responses that pertain to each eigenvalue. Say that by transforming our system's matrices, we force a complex eigenvalue to become real; this will alter the response of the system severely, since a component that was oscillating, will now be either decaying or increasing exponentially, after the transformation.

In SPARSE-SYSID, we propose a *fast* greedy sparsification scheme Iteratively, for all three matrices, we delete small values, while maintaining ther spectrum within ϵ from the one obtained through system identification. Additionally, for $\mathbf{A}$, we also do not allow eigenvalues to switch from complex to real and vice versa. This scheme works very well in practice, providing very sparse matrices, while respecting their spectrum. Doing so is important, because eigenvalues determine the dynamical behavior of our model. In Algorithm 10.1, we provide an outline of the algorithm.

Algorithm 10.1: SPARSE-SYSID: Sparse System Identification of GEBM

```
Input: Training data in the form $\{\mathbf{y}(t), \mathbf{s}(t)\}_{t=1}^{T}$, number of hidden states $n$.
Output: GEBM matrices $\mathbf{A}$ (hidden connectivity matrix), $\mathbf{B}$ (perception matrix), $\mathbf{C}$
    (measurement matrix), and $\mathbf{A}_v$ (voxel-to-voxel matrix).
1: $\{\mathbf{A}^{(0)}, \mathbf{B}^{(0)}, \mathbf{C}^{(0)}\} = \text{SYSID}\left(\{\mathbf{y}(t), \mathbf{s}(t)\}_{t=1}^{T}, n\right)$
2: $\mathbf{A} = \text{EIGENSPARSIFY}(\mathbf{A}^{(0)})$
3: $\mathbf{B} = \text{SINGULARSPARSIFY}(\mathbf{B}^{(0)})$
4: $\mathbf{C} = \text{SINGULARSPARSIFY}(\mathbf{C}^{(0)})$
5: $\mathbf{A}_v = \mathbf{C}\mathbf{A}\mathbf{C}^{\dagger}$
```

10.3.4 Obtaining the voxel-to-voxel connectivity

So far, GEBM as we have described it, is able to give us the hidden functional connectivity and the measurement matrix, but does not directly offer the voxel-to-voxel connectivity, unlike MODEL$_0$, which models it explicitly. However, this is by no means a weakness of GEBM, since there is a simple way to obtain the voxel-to-voxel connectivity (henceforth referred to as $\mathbf{A}_v$) from GEBM's matrices. We highlight the importance of $\mathbf{A}_v$, since this matrix essentially maps abstract latent brain areas to physical brain regions.

Lemma 10.1.:
Assuming that $\mathbf{C}$ is full column rank, the voxel-to-voxel functional connectivity matrix $\mathbf{A}_v$ can be defined and is equal to $\mathbf{A}_v = \mathbf{C}\mathbf{A}\mathbf{C}^{\dagger}$

Algorithm 10.2: EIGENSPARSIFY: Eigenvalue Preserving Sparsification of System Matrix $\mathbf{A}$.

```
Input: Square matrix $\mathbf{A}^{(0)}$.
Output: Sparsified matrix $\mathbf{A}$.
 1: $\boldsymbol{\lambda}^{(0)}$ =EIGENVALUES($\mathbf{A}^{(0)}$)
 2: Initialize $\mathbf{d}_R^{(0)} = \mathbf{0}$, $\mathbf{d}_I^{(0)} = \mathbf{0}$. Vector $\mathbf{d}_R^{(i)}$ holds the element-wise difference of the real
    part of the eigenvalues of $\mathbf{A}^{(i)}$. Similarly for $\mathbf{d}_I^{(i)}$ and the imaginary part.
 3: Set vector c as a boolean vector that indicates whether the $j$-th eigenvalue in $\boldsymbol{\lambda}^{(0)}$ is
    complex or not. One way to do it is to evaluate element-wise the following boolean
    expression: $\mathbf{c} = \big(\text{IMAG}(\boldsymbol{\lambda}^{(0)}) \neq 0\big)$.
 4: Initialize $i = 0$
 5: while $\mathbf{d}_R^{(i)} \leq \epsilon$ and $\mathbf{d}_I^{(i)} \leq \epsilon$ and $\big(\text{IMAG}(\boldsymbol{\lambda}^{(i)}) \neq 0\big) == \mathbf{c}$ do
 6:     Initialize $\mathbf{A}^{(i)} = \mathbf{A}^{(i-1)}$
 7:     $\{v_i^*, v_j^*\} = \arg\min_{v_i,v_j} |\mathbf{A}^{(i-1)}(v_i, v_j)|$
        s.t. $\mathbf{A}^{(i-1)}(v_i, v_j) \neq 0$.
 8:     Set $\mathbf{A}^{(i)}(v_i^*, v_j^*) = 0$
 9:     $\boldsymbol{\lambda}^{(i)}$ =EIGENVALUES($\mathbf{A}^{(i)}$)
10:     $\mathbf{d}_R^{(i)} = |\text{REAL}(\boldsymbol{\lambda}^{(i)}) - \text{REAL}(\boldsymbol{\lambda}^{(i-1)})|$
11:     $\mathbf{d}_I^{(i)} = |\text{IMAG}(\boldsymbol{\lambda}^{(i)}) - \text{IMAG}(\boldsymbol{\lambda}^{(i-1)})|$
12: end while
13: $\mathbf{A} = \mathbf{A}^{(i-1)}$
```

Algorithm 10.3: SINGULARSPARSIFY: Singular Value Preserving Sparsification

```
Input: Matrix $\mathbf{M}^{(0)}$.
Output: Sparsified matrix M
 1: $\boldsymbol{\lambda}^{(0)}$ =SINGULARVALUES($\mathbf{A}^{(0)}$)
 2: Initialize $\mathbf{d}_R^{(0)} = \mathbf{0}$ which holds the element-wise difference of the singular values of
    $\mathbf{A}^{(i)}$.
 3: Initialize $i = 0$
 4: while $\mathbf{d}_R^{(i)} \leq \epsilon$ do
 5:     Initialize $\mathbf{M}^{(i)} = \mathbf{M}^{(i-1)}$
 6:     $\{v_i^*, v_j^*\} = \arg\min_{v_i,v_j} |\mathbf{M}^{(i-1)}(v_i, v_j)|$
        s.t. $\mathbf{M}^{(i-1)}(v_i, v_j) \neq 0$.
 7:     Set $\mathbf{M}^{(i)}(v_i^*, v_j^*) = 0$
 8:     $\boldsymbol{\lambda}^{(i)}$ =SINGULARVALUES($\mathbf{M}^{(i)}$)
 9:     $\mathbf{d}_R^{(i)} = |\boldsymbol{\lambda}^{(i)} - \boldsymbol{\lambda}^{(i-1)}|$
10: end while
11: $\mathbf{M} = \mathbf{M}^{(i-1)}$
```

Proof. The observed voxel vector can be written as

$$\mathbf{y}(t+1) = \mathbf{C}\mathbf{x}(t+1) = \mathbf{C}\mathbf{A}\mathbf{x}(t) + \mathbf{C}\mathbf{B}\mathbf{s}(t)$$

Matrix $\mathbf{C}$ is tall (i.e. $m > n$) and full column rank, thus we can write: $\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) \Leftrightarrow \mathbf{x}(t) = \mathbf{C}^{\dagger}\mathbf{y}(t)$ Consequently, $\mathbf{y}(t+1) = \mathbf{C}\mathbf{A}\mathbf{C}^{\dagger}\mathbf{y}(t) + \mathbf{C}\mathbf{B}\mathbf{s}(t)$ Therefore, it follows that $\mathbf{C}\mathbf{A}\mathbf{C}^{\dagger}$ is the voxel-to-voxel matrix $\mathbf{A}_v$. ■

One of the key concepts behind GEBM is sparsity, which, in the context of the voxel-to-voxel functional connectivity can be interpreted in the same way as in the case of the latent functional connectivity matrix $\mathbf{A}$. However, even though matrices $\mathbf{A}, \mathbf{C}$ are sparse, the product $\mathbf{C}\mathbf{A}\mathbf{C}^{\dagger}$ is not necessarily sparse, since $\mathbf{C}^{\dagger}$ is more dense than $\mathbf{C}$ and therefore the product is dense. In order to obtain a more interpretable, sparse voxel-to-voxel connectivity matrix, we apply the same sparsification technique that we use for the GEBM matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$; since we are not interested in using matrix $\mathbf{A}_v$ in a control system context, but rather as a means of interpreting the derived functional connectivity, we may use Algorithm 10.3 which retains the singular values of the matrix.

There is another, important distinction of GEBM's $\mathbf{A}_v$ from a great amount of prior art which focuses on functional connectivity estimation, where the notion of the functional connectivity is associated with computing a covariance or correlation matrix from the sensor measurements (e.g. [SPL+09]). By doing so, the derived connectivity matrix is *symmetric*, which dictates that the relation between the regions covered by sensors A and B is exactly reciprocal. Our proposed connectivity matrix, on the other hand, as it is evident by the formula $\mathbf{A}_v = \mathbf{C}\mathbf{A}\mathbf{C}^{\dagger}$ is not necessarily symmetric, and it allows for skewed communication patterns between brain regions. As we point out in our discoveries (Section 4), there is potential value in not imposing symmetry constraints.

10.3.5 Connection to Recurrent Neural Networks

In this subsection, we point out an interesting connection of our proposed model GEBM, and a flavor of Artificial Neural Networks, called Recurrent Neural Networks (RNNs). Our proposed model GEBM consists of a layer of *sensor neurons* which transfer the stimulus to the internal, latent neural regions. At the end, there is a *measurement* that transforms the hidden brain activity into the observed signal. In the most popular variant of Artificial Neural Networks (ANNs), the Feed-Forward Neural Networks we encounter a superficially similar structure, where we have an input layer, a hidden layer, and an output layer.

GEBM on the other hand, as also depicted in Fig. 10.3 besides this three-layer layout, allows for connections between the hidden neural regions (depicted by black dots in Fig. 10.3), violating the structure of an FFNN. However, there is a different category of ANNs, the so called Recurrent Neural Networks (RNNs)[Jae02b] which are, in principle, structured exactly like Fig. 10.3; connections between neurons of the hidden layer are allowed, leading to a more expressive model.

We are particularly interested in a specific variant of RNNs, called Echo State Networks (ESNs) [Jae07, Jae01] In ESNs, we have the mapping of an input signal into a set of neurons that comprise the dynamical *reservoir*. There can exist connections between any of these neurons, or *reservoir states*. The output of the reservoir, at the last step, is transformed into the output signal, which is also referred to as *teacher signal*. The parallelism between ESNs and GEBM is as follows: if we consider the *reservoir states* to correspond to the latent neuron regions of GEBM, then the two models look conceptually very similar, in this high level of abstraction.

To further corroborate to the connection of our proposed model GEBM to ESNs, usually the connections between reservoir states are desired to be *sparse*, which is reminiscent of GEBM's specification for $\mathbf{A}$ to be sparse.

In order to formalize the connection between GEBM and ESNs, here we provide the system equations that govern the behavior of an ESN [Jae07]

$$\mathbf{x}(t+1) = f\left(\mathbf{W}\mathbf{x}(t) + \mathbf{W}^{in}\mathbf{u}(t+1) + \mathbf{W}^{fb}\mathbf{y}(t)\right)$$
$$\mathbf{y}(t) = g\left(\mathbf{W}^{out}\left[\mathbf{x}(t) \quad \mathbf{u}(t)\right]\right)$$

where f and g are typically sigmoid functions. Vector $\mathbf{x}(t)$ is the so-called reservoir state, $\mathbf{W}$ is the connectivity between the reservoir states, $\mathbf{W}^{in}$ is the input transformation matrix, $\mathbf{W}^{fb}$ is a feedback matrix, and $\mathbf{W}^{out}$ transforms the hidden reservoir states to the observed output signal of vector $\mathbf{y}(t)$.

If we set f and g to be identity, set $\mathbf{W}^{fb} = \mathbf{0}$, and set the part of $\mathbf{W}^{out}$ that multiplied $\mathbf{u}(t)$ to be zero as well, then the above ESN equations correspond to GEBM.

By pointing out this connection between GEBM and ESNs, we believe that both ends can benefit:

- In this work we introduce a novel sparse system identification algorithm which, as we show in the lines above, is able to solve a particular case of an ESN model very efficiently, thus contributing, indirectly, towards algorithms for training ESNs.
- ESNs usually don't assume linear functions in the system; GEBM does so in the good-enough spirit, however, our intention is to extend GEBM so that it can handle non-linear functions, and capture all the dynamics that our linear assumptions currently fail to. To that end, we can benefit from ESN research (e.g. [Jae02a]).

10.4 Experimental Setup

The code for SPARSE-SYSID has been written in Matlab. For the system identification part, initially we experimented with Matlab's System Identification Toolbox and the algorithms in [Lju99]. These algorithms worked well for smaller to medium scales, but were unable to perform on our full dataset. Thus, in our final implementation, we use the algorithms of [Ver94]. Our code is publicly available at http://www.cs.cmu.edu/~epapalex/src/GeBM.zip. The interested reader can also find experimental evaluation of our proposed algorithm in Section 4 of our KDD paper [PFS+14a].

Dataset Description & Formulation We are using real brain activity data, measured using MEG. MEG (Magnetoencephalography) measures the magnetic field caused by many thousands of neurons firing together, and has good time resolution (1000 Hz) but poor spatial resolution. fMRI (functional Magnetic Resonance Imaging) measures the change in blood oxygenation that results from changes in neural activity, and has good spatial resolution but poor time resolution (0.5-1 Hz). Since we are interested in the temporal dynamics of the brain, we choose to operate on MEG data.

All experiments were conducted at the University of Pittsburgh Medical Center (UPMC) Brain Mapping Center. The MEG machine consists of $m = 306$ sensors, placed uniformly across the subject's scalp. The temporal granularity of the measurements is 5ms, resulting in $T = 340$ time points; after experimenting with different aggregations in the temporal dimension, we decided to use 50ms of time resolution, because this yielded the most interpretable results.

For the experiments, nine *right handed*[2] human subjects were shown a set of 60 concrete English nouns (*apple, knife* etc), and for each noun 20 simple yes/no questions (*Is it edible? Can you buy it?* etc). The subject were asked to press the right button if their answer to each question was 'yes', or the left button if the answer was 'no'. After the subject pressed the button, the stimulus (i.e. the noun) would disappear from the screen. We also record the exact time that the subject pressed the button, relative to the appearance of the stimulus on the screen. A more detailed description of the data can be found in [SPP+12].

In order to bring the above data to the format that our model expects, we make the following design choices: In lack of sensors that measure the response of the eyes to the shown stimuli, we represent each stimulus by a set of semantic features for that specific noun. This set of features is a superset of the 20 questions that we have already mentioned; the value for each feature comes from the answers given by Amazon Mechanical Turk workers. Thus, from time-tick 1 (when the stimulus starts showing), until the button is pressed, all the features that are active for the particular stimulus are set to 1 on our stimulus vector $\mathbf{s}$, and all the rest features are equal to 0; when the button is pressed, all features are zeroed out. On top of the stimulus features, we also have to incorporate the task information in $\mathbf{s}$, i.e. the particular question shown on the screen. In order to do that, we add 20 more rows to the stimulus vector $\mathbf{s}$, each one corresponding to every question/task. At each given experiment, only one of those rows is set to 1 for all time ticks, and all other rows are set to 0. Thus, the number of input sensors in our formulation is $s = 40$ (i.e. 20 neurons for the noun/stimulus and 20 neurons for the task).

As a last step, we have to incorporate the button pressing information to our model; to that end, we add two more voxels to our observed vector $\mathbf{y}$, corresponding to left and right button pressing; initially, those values are set to 0 and as soon as the button is pressed, they are set to 1.

[2]We place emphasis on the right-handedness of the human subjects, because differences in handedness is known cause for inconsistencies in experimental results. To that end, we made sure that all our subjects were right handed.

Finally, we choose $n = 15$ for all the results we show in this section. The results are not very sensitive with respect to small changes in n, thus we chose a relatively small n that yielded interpretable results. In our KDD paper [PFS+14a], we provide insights on how to choose n.

10.5 Discoveries & Discussion

This section is focused on showing different aspects of GEBM at work. In particular, we present the following discoveries:

D1: We provide insights on the obtained functional connectivity from a Neuroscientific point of view.
D2: Given multiple human subjects, we discover regularities and outliers, with respect to functional connectivity.
D3: We demonstrate GEBM's ability to simulate brain activity.
D4: We show how GEBM is able to capture two basic psychological phenomena.

10.5.1 D1: Functional Connectivity Graphs

The primary focus of this work is to estimate the functional connectivity of the human brain, i.e. the interaction pattern of groups of neurons. In the next few lines, we present our findings in a concise way and provide Neuroscientific insights regarding the interaction patterns that GEBM was able to infer.

In order to present our findings, we post-process the results obtained through GEBM in the following way:

We first obtain the MEG-level functional connectivity matrix $\mathbf{A}_v = \mathbf{CAC}^{\dagger}$, and sparsify it, as described in the previous section. As we noted earlier, this matrix is not symmetric, which implies that we may observe skewed information flow patterns between different brain regions. In order to have an estimate of the degree that our matrix is not symmetric, we did the following: We measure the norm ratio

$$r = \frac{\| upper\left(\mathbf{A}_v\right) - lower\left(\mathbf{A}_v\right)^T \|_F}{\|\mathbf{A}_v\|_F}$$

where $upper\,()$ takes the upper triangular part of a matrix, and $lower\,()$ takes the lower triangular part. If $\mathbf{A}_v$ is perfectly symmetric, then $r = 0$, since the upper and lower triangular parts would be equal. However, in our case, $r = 0.6$, indicating that there is a considerable amount of skew in the communication between MEG sensors.

The data we collect come from 306 sensors, placed on the human scalp in a uniform fashion, and the connectivity between those sensors is encoded in matrix $\mathbf{A}_v$. Each of those 306 sensors is measuring activity from one of the four main regions of the brain, i.e.

- **Frontal Lobe**, associated with attention, short memory, and planning.
- **Parietal Lobe**, associated with movement.

- **Occipital Lobe**, associated with vision.
- **Temporal Lobe**, associated with sensory input processing, language comprehension, and visual memory retention.

Even though our sensors offer within-region resolution, for exposition purposes, we chose to aggregate our findings per region; by doing so, we are still able to provide useful neuroscientific insights.

Figure 10.4 shows the functional connectivity graph obtained using GeBM. The weights indicate the strength of the interaction, measured by the number of distinct connections we identified. These results are consistent with current research regarding the nature of language processing in the brain. For example, Hickock and Poeppel [HP04] have proposed a model of language comprehension that includes a “dorsal” and “ventral” pathway. The ventral pathway takes the input stimuli (spoken language in the case of Hickock and Poeppel, images and words in ours) and sends the information to the temporal lobe for semantic processing. Because the occipital cortex is responsible for the low level processing of visual stimuli (including words) it is reasonable to see a strong set of connections between the occipital and temporal lobes. The dorsal pathway sends processed sensory input through the parietal and frontal lobes where they are processed for planning and action purposes. The task performed during the collection of our MEG data required that subjects consider the meaning of the word in the context of a semantic question. This task would require the recruitment of the dorsal pathway (occipital-parietal and parietal-frontal connections). In addition, frontal involvement is indicated when the task performed by the subject requires the selection of semantic information [BD11], as in our question answering paradigm. It is interesting that the number of connections from parietal to occipital cortex is larger than from occipital to parietal, considering the flow of information is likely occipital to parietal. This could, however, be indicative of what is termed “top down” processing, wherein higher level cognitive processes can work to focus upstream sensory processes. Perhaps the semantic task causes the subjects to focus in anticipation of the upcoming word while keeping the semantic question in mind. It is important to note here that we were able to notice this “top down” processing pattern, thanks to fact that GeBM does not impose symmetry constraints in the connectivity matrix.

10.5.2 D2: Cross-subject Analysis

In our experiments, we have 9 participants, all of whom have undergone the same procedure, being presented with the same stimuli, and asked to carry out the same tasks. Availability of such a rich, multi-subject dataset inevitably begs the following question: are there any differences across people’s functional connectivity? Or is everyone, more or less, wired equally, at least with respect to the stimuli and tasks at hand?

By using GeBM, we are able (to the extent that our model is able to explain) to answer the above question. We trained GeBM for each of the 9 human subjects, using the entire data from all stimuli and tasks, and obtained matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$ for each person. For the purposes of answering the above question, it suffices to look at matrix $\mathbf{A}$ (which is the

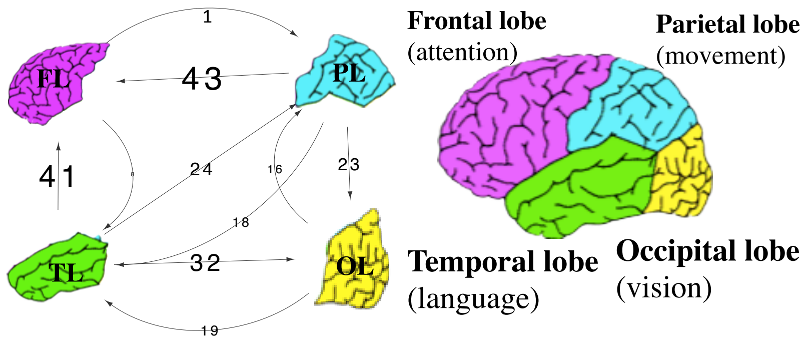


Figure 10.4: **The functional connectivity derived from GeBM**. The weights on the edges indicate the number of inferred connections. Our results are consistent with research that investigates natural language processing in the brain.

hidden functional connectivity), since it dictates the temporal dynamics of the brain activity. At this point, we have to note that the exact location of each sensor can differ between human subjects, however, we assume that this difference is negligible, given the current voxel granularity dictated by the number of sensors.

In this multi-subject study we have two very important findings:

- **Regularities**: For 8 out of 9 human subjects, we identified almost identical GeBM instances, both with respect to RMSE and to spectrum. In other words, for 8 out of 9 subjects in our study, the inferred functional connectivity behaves almost identically. This fact most likely implies that for the particular set of stimuli and assorted tasks, the human brain behaves similarly across people.
- **Anomaly**: One of our human subjects (#3) deviates from the aforementioned regular behavior.

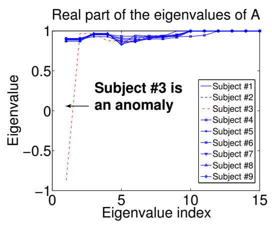
In Fig. 10.5(a) & (b) we show the real and imaginary parts of the eigenvalues of $\mathbf{A}$. We can see that for 8 human subjects, the eigenvalues are almost identical. This finding agrees with neuroscientific results on different experimental settings [TV12], further demonstrating GeBM's ability to provide useful insights on multi-subject experiments. For subject #3 there is a deviation on the real part of the first eigenvalue, as well as a slightly deviating pattern on the imaginary parts of its eigenvalues. Figures 10.5(c) & (d) compare matrix $\mathbf{A}$ for subjects 1 and 3. There is a unique fact pertaining to Subject #3's connectivity matrix: there is a negative value on the diagonal (blue square at the $(8,8)$ entry), in other words a negative self-loop in the connectivity graph, which causes the differences in the behavior of Subject #3's system.

According to Neuroscientific studies (e.g. [HZJ+06]), the first "culprit" for causing such a discrepancy in the estimated model would be the handedness of Subject 3. However, as we highlighted in the beginning of the section, all 9 human subjects were *right handed*, and thus, difference in handedness cannot be a plausible explanation for this anomaly.

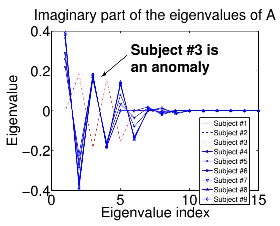
We subsequently turned our attention to the conditions under which the experiment was conducted, and according to the person responsible for the data collection of Subject #3:

> *There was a big demonstration outside the UPMC building during the scan, and I remember the subject complaining during one of the breaks that he could hear the crowd shouting through the walls.*

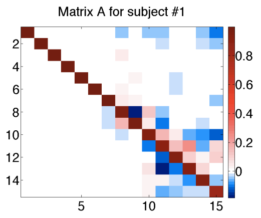
This is a plausible explanation for the deviation of GEBM for Subject #3.



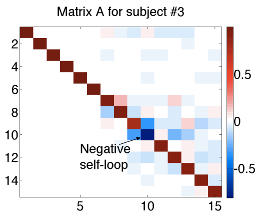
(a) Real part of eigenvalues



(b) Imaginary part of eigenvalues



(c) Subject #1



(d) Subject #3

Figure 10.5: **Multi-subject analysis**: Sub-figures (a) and (b), show the real and imaginary parts of the eigenvalues of matrix $\mathbf{A}$ for each subject. For all subjects but one (subject #3) the eigenvalues are almost identical, implying that the GEBM that captures their brain activity behaves more or less in the same way. Subject #3 on the other hand is an outlier; indeed, during the experiment, the subject complained that he was able to hear a demonstration happening outside of the laboratory, rendering the experimental task assigned to the subject more difficult than it was supposed to be. Sub-figures (c) and (d) show matrices $\mathbf{A}$ for subject #1 and #3. Subject #3's matrix seems sparser and most importantly, we can see that there is a negative entry on the diagonal, a fact unique to subject #3.

10.5.3 D3: Brain Activity Simulation

An additional way to gain confidence on our model is to assess its ability to simulate/predict brain activity, given the inferred functional connectivity. In order to do so, we trained GEBM using data from all but one of the words, and then we simulated brain activity time-series for the left-out word. In lieu of competing methods, we compare our proposed method GEBM against our initial approach (whose unsuitability we have argued for in Section 10.3, but we use here in order to further solidify our case). As an initial state for GEBM, we use $\mathbf{C}^{\dagger}\mathbf{y}(0)$, and for MODEL$_0$, we simply use $\mathbf{y}(0)$. The final time-series we show, both for the real data and the estimated ones are normalized to unit norm, and plotted in absolute values. For exposition purposes, we sorted the voxels according to the ℓ_2 norm of their time series vector, and we are displaying the high ranking ones (however, the same pattern holds for all voxels)

In Fig. 10.6 we illustrate the simulated brain activity of GEBM (solid red), compared against the ones of MODEL$_0$ (using LS (dash-dot magenta) and CCA (dashed black)), as well as the original brain activity time series (dashed blue) for the four highest ranking voxels. Clearly, the activity generated using GEBM is far more realistic than the results of MODEL$_0$.

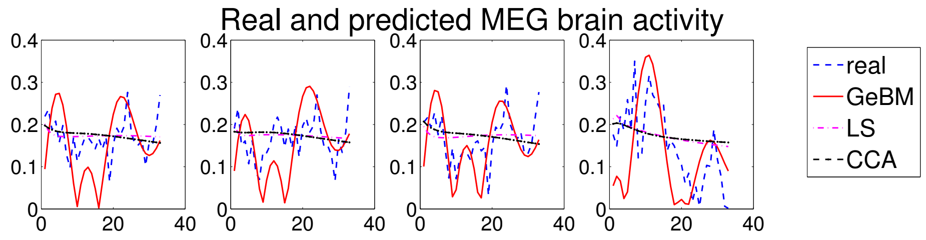


Figure 10.6: **Effective brain activity simulation**: Comparison of he real brain activity and the simulated ones using GEBM and MODEL$_0$, for the first four high ranking voxels (in the ℓ_2 norm sense).

10.5.4 D4: Explanation of Psychological Phenomena

As we briefly mentioned in the Introduction, we would like our proposed method to be able to capture some of the psychological phenomena that the human brain exhibits. We by no means claim that GEBM is able to capture convoluted still little-understood physiological phenomena , however, in this section we demonstrate GEBM's ability to simulate two very basic phenomena, *habituation* and *priming*. Unlike the previous discoveries, the following experiments are on synthetic data and their purpose is to showcase GEBM's additional strengths.

Habituation In our simplified version of habituation, we observe the demand behaviour: Given a repeated stimulus, the neurons initially get activated, but their activation levels decline ($t = 60$ in Fig. 10.7) if the stimulus persists for a long time ($t = 80$ in Fig. 10.7). In Fig. 10.7, we show that GEBM is able to capture such behavior. In particular, we show the desired input and output for a few (observed) voxels, and we show, given the functional connectivity obtained according to GEBM, the simulated output, which exhibits the same, desired behavior. In order to simplify the training data generation, we produce an instantaneous "switch-off" of the activity in the desired output, which is a crude approximation of an gradual attenuation, which is expected. Remarkably, though, GEBM (using Algorithm 10.1) is able to produce a very realistic, gradually attenuating activity curve.

Priming In our simplified model on priming, first we give the stimulus *apple*, which sets off neurons that are associated with the fruit 'apple', as well as neurons that are associated with Apple Inc. . Subsequently, we are showing a stimulus such as *iPod*; this predisposes the regions of the brain that are associated with Apple inc. to display some small level of activation, whereas suppressing the regions of the brain that are associate with apple (the fruit). Later on, the stimulus *apple* is repeated, which, given the aforementioned predisposition, activates the voxels associated with Apple (company) and suppresses the ones associated with the homonymous fruit.

Figure 10.8 displays is a pictorial description of the above example of priming; given

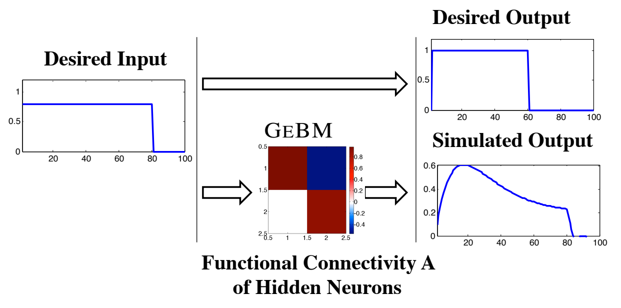


Figure 10.7: **GeBM captures *Habituation***: Given repeated exposure to a stimulus, the brain activity starts to fade.

desired input/output training pairs, we derive a model that obeys GeBM using our proposed Algorithm 10.1, such that we match the priming behavior.

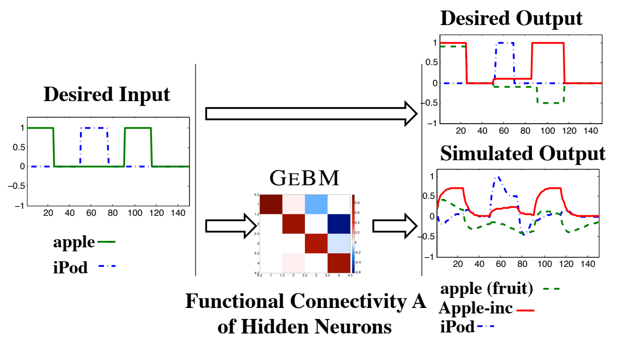


Figure 10.8: **GeBM captures *Priming***: When first shown the stimulus *apple*, both neurons associated with the fruit *apple* and *Apple Inc.* get activated. When showing the stimulus *iPod* and then *apple*, *iPod* predisposes the neurons associated with Apple inc. to get activated more quickly, while suppressing the ones associated with the fruit.

10.6 Related Work

Brain Functional Connectivity Estimating the brain's functional connectivity is an active field of study of computational neuroscience. Examples of works can be found in [Sak11, GKRM03, FFDM12]. There have been a few works in the data mining community as well: In [SPL+09], the authors derive the brain region connections for Alzheimer's patients, and recently [DGCW13] that leverages tensor decomposition in order to discover the underlying network of the human brain. Most related to the present work is the work of Valdes et al [VSSBLC+05], wherein the authors propose an autoregressive model (similar to Model$_0$) and solve it using regularized regression. However, to the best of our knowledge, this work is the first to apply system identification concepts to this problem.

Psychological Phenomena A concise overview of literature pertaining to habitutation can be found in [TS66]. A more recent study on habitutation can be found in [RAB+09].

The definition of priming, as we describe it in the lines above concurs with the definition found in [FSF99]. Additionally, in [PB98], the authors conduct a study on the effects of priming when the human subjects were asked to write sentences. The above concepts of priming and habituation have been also studied in the context of spreading activation [And83, CL75] which is a model of the cognitive process of memory.

Control Theory & System Identification System Identification is a field of control theory. In the appendix we provide more theoretical details on subspace system identification, however, [Lju99] and [VV07] are the most prominent sources for system identification algorithms.

Network Discovery from Time Series Our work touches upon discovering underlying network structures from time series data; an exemplary work related to the present chapter is [VVA+06] where the authors derive a who-calls-whom network from VoIP packet transmission time series.

10.7 Conclusions

In this chapter, we make the following contributions:

- **Analytical model & Algorithm** : We propose GEBM, a novel model of the human brain functional connectivity. We also introduce SPARSE-SYSID, a novel sparse system identification algorithm that estimates GEBM
- **Effectiveness**: GEBM simulates psychological phenomena (such as habituation and priming), as well as provides valuable neuroscientific insights.
- **Validation**: We validate our approach on real data, where our model produces brain activity patterns, remarkably similar to the true ones.
- **Multi-subject analysis**: We analyze measurements from 9 human subjects, identifying a consistent connectivity among 8 of them; we successfully identify an outlier, whose experimental procedure was compromised.
- **Cross-disciplinary connections**: We highlight connections between disparate areas: 1) Neuroscience, 2) Control Theory & System Identification, and 3) Psychology, and we provide insights on the relation of GEBM to Recurrent Neural Networks.

The problem of identifying the functional connectivity of the human brain is very important in the field of Neuroscience, as it contributes to our knowledge about how the human brain operates and processes information. However, solving this problem can have applications that go far beyond Neuroscience:

- **Improving Artificial Intelligence & Machine Learning**: Having a better understanding of how the human brain processes information is of paramount importance in the design of intelligent algorithms; in particular, such understanding can be greatly beneficial especially to fields that are inspired by the way that the human brain operates, with a prime example of Deep Learning [Sch14], which is gaining increasing attention.

- **Detecting & ameliorating learning disorders**: On a different spin, understanding the functional connectivity of the brain can be used in order to detect learning disorders in children, at a very early stage, in a non-invasive way. If we have a model for the functional connectivity, as well as an understanding of how different learning disorders may perturb that model, we might be able to, first, detect the disorder, and subsequently, by targeting the child's education and closely monitoring changes in the connectivity, we may be able to ameliorate the effects of certain disorders.

Part IV

Applications: Social Networks and the Web

Chapter based on material published in [PAI13].

Chapter 11

Do more Views of a Graph help? Community Detection and Clustering in Multi-View Graphs

Using different views of a social network results in more accurate community detection.

Given a network with multiple types (or views) of edges (e.g., collaboration, citation, friendship), can community detection and graph clustering benefit? In this Chapter, we propose MULTICLUS and GRAPHFUSE, two *multi-graph* clustering techniques powered by Minimum Description Length and Tensor analysis, respectively. Our results on real and synthetic data demonstrate higher clustering accuracy than state-of-the-art baselines that do not exploit the multiple views of our data.

11.1 Introduction

Many data types, nowadays, can be represented by a network in which entities correspond to nodes and relationships between entities correspond to edges between nodes. However, as the data complexity increases the standard definition of a simple graph falls short to represent the complex semantics that reside in real world networks. More specifically, we can have multiple sources of information describing different types of relationships associated with the nodes in a network. For example, a set of users may have various communication channels (e.g. phone, email, messaging, etc.) or researchers in a field may have different dimensions of interaction (co-authorship, citations, using similar keywords). As a result different information networks, involving the same set of objects, can be inferred. In both scenarios users (researchers) are the nodes of the network while each relation (dimensions of interaction) represents a different semantic relationship between two objects in the graph. These multiple semantics (or dimensions) cannot be described with only one simple graph but they may be expressed by a set of different graphs sharing the same set of nodes. These multi-source networks are often referred to as multi-view graphs, multi-dimensional graphs, multi-layer graphs, or simply multi-graphs[TWL12].

Due to the popularity of networks, mining network patterns has become an important task in different domains such as computer science, physics, economy, sociology, biology, and chemistry. One of the most important and challenging research problems that attracts much attention is graph clustering [HSH$^+$10, XYFS07, XKW$^+$12]. The goal of this task is to obtain groups of nodes that are similar w.r.t. some structural or node attribute information. Many approaches were proposed in the context of single graph clustering[AW10, LLM10, vL07] while the problem of clustering multi-dimensional graphs has gained interest only recently[TLD09].

Multi-graph clustering aims to fully exploit the interactions among different dimensions of a given network and is able to take into account the correlations among them, whereas standard approaches that manage each graph independently cannot leverage the correlated information coming from the different dimensions. Moreover, information coming from multiple sources may have different characteristics and value. For example, the citation information among papers is highly valuable for clustering, however it may be quite sparse. On the other hand, the co-term information are plenty, however it may be noisy as two papers having similar terms is not directly indicative that they belong to the same topic (e.g., the term *cluster* in the data mining field or in the cloud computing area). The motivation behind multi-graph clustering is exactly to combine and blend in informative-but-sparse and plenty-but-noisy information holistically to strengthen each other and improve the clustering performance.

In this Chapter we propose two new methods for clustering multi-view graphs:

1. Our first proposed method, MULTICLUS, is based on an information theoretical approach, where the formulation aims to simultaneously "describe" all the views of the network using as few bits as possible. The developed algorithm strives to find the clustering that can best compress the multi-graph at all views. The advantage of our first method is that it requires no user-defined parameters, i.e. can determine the number of clusters automatically. On the other hand, it can work only with binary, i.e. unweighted, graphs.
2. Our second proposed method, GRAPHFUSE, is based on a tensor factorization approach, which can handle weighted graphs and uses search heuristics to find the best number of clusters. The code for GRAPHFUSE can be found at `http://www.cs.cmu.edu/~epapalex/src/GraphFuse.zip`.

We compare our methods against two baseline strategies on both synthetic and real-world multi-graphs, and show that MULTICLUS and GRAPHFUSE yield superior performance over competitors in all clustering tasks.

The rest of this paper: related work on multi-graph clustering (Sec. 12.2), our problem formulation (Sec. 11.3), proposed methods (Sec. 11.4 & Sec. 11.5), quantitative and qualitative results on both synthetic and real data (Sec. 11.6), and concluding remarks (Sec. 11.7).

11.2 Related Work

Multi-dimensional networks allow for the representation of complex data with different semantic relations between objects. For instance in the context of social network analysis, [BCG11] introduces the problem of community detection over multi-dimensional graphs. The authors model the different relationships between two nodes using different types of edges. Based on this model, they introduce a new community detection algorithm.

In [TWL12], the authors perform interaction analysis among communities over heterogeneous multi-dimensional social networks like Del.icio.us, Flickr, and YouTube. They show the usefulness of this rich representation to model real complex interaction between users. In order to extract interaction behavior from multi-graph data, [BGHS12] presents a graph mining approach to extract quasi-clique structures from multi-dimensional graphs. More specifically the work is devoted to extract multi-dimensional coherent quasi-cliques which define clusters of vertices that are densely connected by edges belonging to the same dimension.

A first approach that cope with the issue of clustering multi-dimensional networks is proposed in [ZB07]. In this work a generalization of normalized cut for multi-dimensional graphs is developed. The framework leads to a mixture of Markov chains defined over each dimension of the multi-dimensional graph. In [RP11] a meta-clustering that deals with multi-dimensional networks is introduced. They do not focus on a specific clustering algorithm that directly deals with the multi-dimensionality, on the contrary they introduce a meta strategy that aggregates the independent clusterings derived by the different dimensions.

In [TLD09] the authors propose a factorization method based on linked matrices to solve the multi-graph clustering problem. In this model, each graph is approximated by a graph-specific factor with a common factor shared by all the graphs. This common factor is used as a link among the different dimensions. In [SM12a] a new variational Bayesian framework for clustering multi-graphs is proposed. This approach is based on a probabilistic generative model based on variational Bayesian estimation. The algorithm is mainly tested over biological networks in which different interaction networks associated with the same set of genes are built. As the method is based on a generative model, the approach requires extra parameters (e.g., hyper-parameter of the Dirichlet distribution).

Our proposed work, in contrast to heuristic approaches, is based on theoretical foundations of information theory and tensor decompositions, and is perfectly suitable for 3-mode multi-graph data (nodes $\times$ nodes $\times$ views).

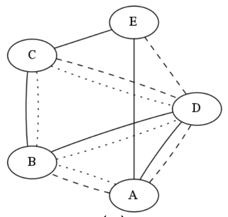
11.3 Problem Definition

A multi-graph $\mathcal{G}$ is a set of m graphs defined over the same set of nodes. More formally, $\mathcal{G} = \{G_l\}_{l=1}^{m}$ where each graph $G_l = (V, E_l)$ consists of the set of nodes V and a set of edges $E_l : V \times V$. n denotes the number of nodes $|V|$.

In Fig. 11.1(a) a simple example is shown. The multi-graph in (a) is defined over 5 nodes $V = \{\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}, \mathbf{E}\}$ connected by 3 different types of edges represented by *solid*, *dotted*, and *dashed* lines. Each dimension represents one of the 3 different edge-semantics and can be associated with a standard adjacency matrix (Fig. 11.1(b), 11.1(c) and 11.1(d)).

Given the above notation, the multi-dimensional graph clustering problem can be stated as follows: Given a multi-graph $\mathcal{G}$, find a partitioning C of the nodes in V such that $\forall_{C_i, C_j \in C} C_i \cap C_j = \emptyset$, and $\bigcup_i C_i = V$. The primary goal of the partitioning is to often optimize an objective function that aims to minimize inter-cluster cross-edges while yielding well-connected dense clusters with high intra-cluster connectivity, at all graph dimensions.

In this work, we formulate (1) a description-length-based, and (2) a tensor-decomposition-based objective function to address this goal. We describe our proposed solutions in detail next.



(a)

	A	**B**	**C**	**D**	**E**
A	0	0	0	0	1
B	0	0	1	1	0
C	0	1	0	0	1
D	0	1	0	0	0
E	1	0	1	0	0

(b)

	A	**B**	**C**	**D**	**E**
A	0	1	0	1	0
B	1	0	0	0	0
C	0	0	0	1	0
D	1	0	1	0	1
E	0	0	0	1	0

(c)

	A	**B**	**C**	**D**	**E**
A	0	0	0	0	0
B	0	0	1	1	0
C	0	1	0	1	0
D	0	1	1	0	0
E	0	0	0	0	0

(d)

Figure 11.1: (a) Example multi-dimensional graph. Each different dimension (i.e. edge-type) is represented as a matrix: solid (-) (b), dashed (- -) (c) and dotted (..) edges (d).

11.4 Our First Attempt: MULTICLUS

As a first attempt for multi-graph clustering, we generalize earlier work on automatic cross-associations [Cha04] so that it can handle multiple graphs at the same time. [ATMF12] recently extended cross-associations to attributed graphs. Following similar ideas, we formulate the problem as a data compression task for multiple adjacency matrices, each defined w.r.t. a different view of the network.

More specifically, our solution is based on the Minimum Description Length (MDL) principle [Ris83]. That is, we formulate an objective function based on the total number of bits required to "describe" the multiple adjacency matrices based on a common clustering. We define our formulation in detail next.

11.4.1 Objective Function Formulation

Simply put, MDL is a model selection principle which is based on lossless compression. When regarded as encoding the data by a "sender" to describe it to a "receiver", the formulation consists of (1) model, and (2) data description given the model, such that the "receiver" could fully decode the original data. The goal is to use as few bits as possible such that the description cost is minimized.

As we are dealing with graph clustering, our models consist of a set of possible clusterings. In addition, our data consists of "blocks" in each input adjacency matrix, defined by a given clustering that is the same for all matrices. Our goal then is to find *the* clustering that would minimize the total model and data description cost (in bits). We explain each description cost below.

Model Description Cost consists of encoding the number of node clusters as well as the corresponding assignment of nodes to their respective clusters:

- The number of nodes n requires $\log^* n$ bits, where $\log^*$ is the universal code length for integers [Ris83].
- The number of node clusters k requires $\log^* k$ bits.
- The node cluster assignments with arithmetic coding requires $nH(P)$ bits, where H denotes the Shannon entropy function, P is a multinomial random variable with the probability $P_i = \frac{r_i}{n}$ and r_i is the size of the ith node cluster, $1 \geq i \geq k$. The cluster assignments for all views of the network is the same (shared clustering), and will be described once.

Data Description Cost consists of encoding the matrix blocks, for each G_l:

- For each block $B^l_{i,j}$, $i, j = 1, ..., k$ and $l = 1, ..., m$, $n_1(B^l_{ij})$ is the number of 1s in the sub-matrix, which requires $\log^* n_1(B^l_{ij})$ bits.
- Having encoded the summary information about the rectangular blocks, we next encode the actual blocks B^l_{ij}. We can calculate the density $P_{ij}(1)$ of 1s in B^l_{ij} using the description code above as $P_{ij}(1) = \frac{n_1(B^l_{ij})}{n(B^l_{ij})}$, where $n(B^l_{ij}) = n_1(B^l_{ij}) + n_0(B^l_{ij}) = r_i \times r_j$, where $n_0(B^l_{ij})$ and $n_1(B^l_{ij})$ are the number of 0s and 1s in B^l_{ij}, respectively. Then the number of bits required to encode each block using arithmetic coding is: $E(B^l_{ij}) = -n_1(B^l_{ij}) \log_2(P_{ij}(1)) - n_0(B^l_{ij}) \log_2(P_{ij}(0)) = n(B^l_{ij}) H(P_{ij})$.

11.4.2 Our Objective Function: Total Encoding Cost (length in bits)

$$MDL_{objFunc} = \log^* n + \log^* k - \sum_{i=1}^{k} r_i \log_2(\frac{r_i}{n}) + \sum_{l=1}^{m}\sum_{i=1}^{k}\sum_{j=1}^{k} \log^* n_1(B_{ij}^l) + E(B_{ij}^l)$$

Our objective function defines the total description, i.e. encoding, cost of our input multiple graphs *given* a clustering. Our aim is to use an algorithm that will find the clustering that *minimizes* the total cost. Finding the optimal clustering with the minimum cost has been stated to be NP-hard in [SFPY07]. Therefore, we resort to a heuristic iterative algorithm based on a top-down clustering approach as in [Cha04], with an extension to consider total encoding cost over all input matrices. The main idea is to iteratively increase the number of clusters and reassign each row and column, i.e. node, to the cluster for which the reduction in total cost is the most. As such, the algorithm is greedy and monotonic and often converges to a local optimum. In practice, however, it has been shown to perform quite well on both synthetic and real-world graphs. We refer to [ATMF12, Cha04, SFPY07] for more details.

11.5 Proposed Method: GraphFuse

In this section, we introduce GraphFuse, a method that treats all different views of a multi-graph as a *tensor*; more specifically, we consider the adjacency matrix of each view of the graph as a different slice of a three-way tensor, and we introduce a method that is able to cluster the given graph, and additionally, identify the influence of each view on each cluster extracted.

Notation A scalar is denoted by a lowercase, italic letter, e.g. x. A column vector is denoted by a lowercase, boldface letter, e.g. $\mathbf{x}$. A matrix is denoted by an uppercase, boldface letter, e.g. $\mathbf{X}$. A tensor is denoted by an uppercase, boldface, underlined letter, e.g. $\underline{\mathbf{X}}$.

Brief introduction to tensors & tensor decompositions. An n-mode tensor is essentially a multidimensional matrix, indexed by n variables. In this work, we focus on three-way tensors, due to their immediate application to the concept of graphs with multiple views. Namely, each slice of a tensor can be viewed as the adjacency matrix of a different view of a particular graph.

Parafac decomposition [Har70] is a highly popular method for tensor analysis. Specifically, the Parafac decomposition of a tensor $\underline{\mathbf{X}}$ into F rank-one components is $\underline{\mathbf{X}} \approx \sum_{f=1}^{F} \mathbf{a}_f \circ \mathbf{b}_f \circ \mathbf{c}_f$, where $\mathbf{a} \circ \mathbf{b} \circ \mathbf{c}(i,j,k) = \mathbf{a}(i)\mathbf{b}(j)\mathbf{c}(k)$.

Recently, Parafac-Slf [PS11] (SLF stands for sparse latent factors), a variation of the Parafac decomposition was proposed. This decomposition imposes sparsity constraints on the latent factors of the plain Parafac decomposition, i.e. the columns of matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$. By imposing sparsity, essentially one is able to do tensor co-clustering: The non-zeros of the i-th column of $\mathbf{A}$ select which elements of the first mode of the tensor belong to the i-th co-cluster, and so on. For a detailed survey of tensors and tensor decompositions, see [KB09].

Description of GRAPHFUSE. In this section, we introduce our tensor based approach for multi-graph clustering, called GRAPHFUSE. At the heart of our proposed method lies the PARAFAC decomposition with Sparse Latent Factors (SLF) [PS11], which we described above shortly.

PARAFAC-SLF is specifically tailored to *soft* co-clustering, in which we seek to find (possibly overlapping) subsets of rows, columns, and fibers of a given tensor, possibly ignoring some "noisy" data. On the contrary, for the task at hand, we need to assign every node of the graph to one of the available clusters. This task definition, at first, makes the direct application of PARAFAC-SLF to the problem seem inappropriate. However, in this section, we introduce a few modifications to PARAFAC-SLF, in order to make it suitable for *hard* multi-graph clustering. The modifications we applied are the following:

1) PARAFAC-SLF allows, by definition, overlapping, i.e. one row, column, or fiber of the tensor to belong to more than one cluster. For the first two modes of the tensor, which correspond to the nodes of the graph, this overlapping freedom needs to be restricted, such that a node belongs to *at most* one cluster. In order to do that, for each node that belongs to more than one clusters, we assign it to the one with the higher weight, i.e. retain the maximum element of each row of $\mathbf{A}$ and $\mathbf{B}$ (line 6 of Algorithm 11.1). We do not need to do the same for matrix $\mathbf{C}$, since it captures the influence of each graph view on each cluster, and we ideally *require* overlapping effects in this context.

2) Additionally, with PARAFAC-SLF being a *soft* technique, some nodes might have been completely ignored in the result, as they may exhibit very low variation (and usually being ultimately noise). In this setting, however, every node has to be assigned to *exactly* one cluster. To this end, we first extract $R-1$ components (line 1 of Algorithm 11.1). This means that one arbitrary node of the graph either belongs to one of those $R-1$ components (i.e. clusters) or is not assigned anywhere. If the latter occurs, we create an R-th cluster, in which all "left-out" nodes are assigned.

In Algorithm 11.1 we provide the pseudo-code of our proposed algorithm, as thoroughly described in the previous lines. Vectors $\boldsymbol{\alpha}_I$ and $\boldsymbol{\alpha}_J$ indicate the clusters dictated by the first and the second modes of the tensor respectively. If the graph is undirected, then the two clustering results should be similar, if not identical, but if the tensor captures non-reciprocal relations, then it is natural to expect variations between $\boldsymbol{\alpha}_I$ and $\boldsymbol{\alpha}_J$.

Connection of GRAPHFUSE to LMF [TLD09]. One of the recent existing approaches to multi-graph clustering is introduced in [TLD09], where the authors propose a Linked Matrix Factorization (LMF) model; this approach approximates every view $\mathbf{X}_k$ of the graph as $\mathbf{X}_k \approx \mathbf{P}\boldsymbol{\Lambda}_k\mathbf{P}^T$ with Frobenius norm regularization on both $\boldsymbol{\Lambda}_k$ and $\mathbf{P}$. In this work, we show that LMF may, under certain conditions, be expressed as a tensor decomposition which bears certain similarities to our approach but is differentiated in some key points. Nevertheless, it is still of interest to investigate the theoretical similarities of the two approaches.

Algorithm 11.1: GRAPHFUSE

```
Input: Multi-graph $\mathcal{G}$ in tensor form $\underline{\mathbf{X}}$ of size $I \times J \times K$, number of clusters $R$, sparsity penalty factor $\lambda$.
Output: Assigments to clusters $\boldsymbol{\alpha}_I$ and $\boldsymbol{\alpha}_J$. Matrix $\mathbf{C}$ of size $K \times R$ that shows the contribution of each
    one of the $K$ views to each one of the $R$ clusters.
1: $\{\mathbf{A}, \mathbf{B}, \mathbf{C}\}$ = PARAFAC SLF ($\underline{\mathbf{X}}$, $R-1$, $\lambda$).
2: for $i = 1 \cdots I$ do
3:     if $\mathbf{A}(i,:) = \mathbf{0}$ then
4:         $\boldsymbol{\alpha}_I(i) = R$
5:     else
6:         $\boldsymbol{\alpha}_I(i) = \arg\max \mathbf{A}(i,:)$
7:     end if
8: end for
9: Repeat iteration 2-8 for all $J$ rows of $\mathbf{B}$. Labels are output in $\boldsymbol{\alpha}_J$.
```

Lemma 11.1.:
If matrices $\mathbf{\Lambda}_k$ of LMF are diagonal, then the LMF model can be expressed as a regularized *symmetric* (in the first two modes) PARAFAC, or regularized INDSCAL [KB09] model.

Proof. The INDSCAL decomposition is simply a PARAFAC decomposition in which the matrices $\mathbf{A}$ and $\mathbf{B}$ are identical. For simplicity, we drop the regularization terms from all the equations discussed. If we express the k-th slice of the tensor as $\mathbf{X}_k$, then for INDSCAL we can write $\mathbf{X}_k \approx \mathbf{A}\text{diag}(\mathbf{C}(k,:))\mathbf{A}^T$ (where diag$(\mathbf{C}(k,:))$ creates a diagonal matrix using the k-th row of $\mathbf{C}$). Similarly, LMF approximates each graph view (or slice of the tensor) as $\mathbf{X}_k \approx \mathbf{P}\mathbf{\Lambda}_k\mathbf{P}^T$. If we rename $\mathbf{A}$ to $\mathbf{P}$ (simply a variable substitution), and in the case of matrices $\mathbf{\Lambda}_k$ being diagonal, the optimal solution of the two models is concluded to be the same. ■

Our approach, however, is based on an improvement of the PARAFAC model, which, by imposing sparsity promoting constraints, is able to perform better in terms of clustering quality, thus differentiating itself from the aforementioned models. In [TLD09], the authors do not specify how often the $\mathbf{\Lambda}_k$ matrices are indeed diagonal, however, we deem interesting to point out this connection.

11.6 Experiments

In this section, we provide both quantitative and qualitative results for our proposed algorithms. For the quantitative evaluation, we compare the clustering quality of our approaches to that of two widely used baselines which do not take advantage of the multi-view nature of the data. To this end, we use 5 different datasets (3 synthetic, 2 real) which we describe next.

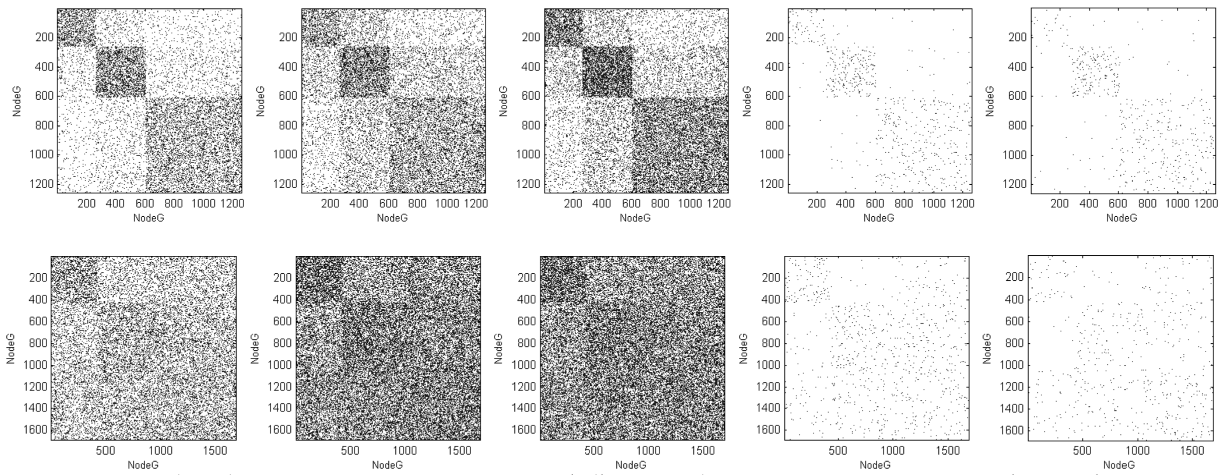


Figure 11.2: (top) SYNTHETIC-2 SIM and (bottom) SYNTHETIC-3 DIF share the same clustering scheme, with different amount of cross edges and cluster densities. DIF multi-graph, by construction, is harder to cluster than SIM.

11.6.1 Data description

Synthetic data generation.

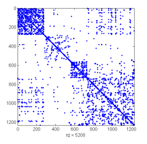
In order to study the performance of our algorithm on different types of graphs, we generated synthetic multi-graphs with various number of views, containing various density views, and with varying clustering quality. Our generative algorithm is based on the planted partitions model [CK01]. Simply put, given the desired number of nodes in each cluster we split the adjacency matrix into blocks defined by the partitioning. For each block B_{ij}, the user also provides a probability p_{ij}. Using a random process we assign a 1, i.e. an edge, for each possible entry in the block, and 0 otherwise. In other words, p_{ij} specifies the density of each block. We also add noise to all graphs; we use $p_{noise} = 0.05$ in our experiments.

Using the planted partitions model, we generated 3 different multi-graphs. In SYNTHETIC-1, we have 5 views and 5 clusters of various sizes. In view 1, clusters are dense with few cross edges, in views 2-3 the clusters are dense with many cross edges, and the views 4-5 have very sparse clusters with also sparse cross edges. Due to limited space, we show the spy-plots for the views of only SYNTHETIC-2 and SYNTHETIC-3 in Fig. 11.2. Notice that these two synthetic multi-graphs each have 5 views and 3 clusters, and they share the same clustering. The difference between them is the amount of cross edges, or noise, introduced. By construction, clustering SYNTHETIC-3 is expected to be harder; hence we refer to these multi-graphs as SIM for simple and DIF for difficult to cluster, respectively.

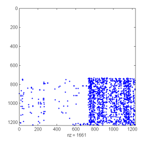
Real data description.

The two real datasets, DBLP-1 and DBLP-2 come from the DBLP online database[1]. More specifically, each of the views of these two datasets corresponds to an author-author graph. In the first view, each edge represents a citation from one author to the other.

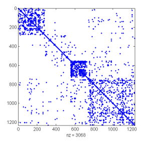
[1]http://dblp.uni-trier.de/



(a) citation

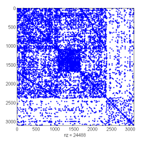


(b) co-auth.

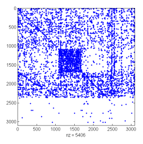


(c) co-term

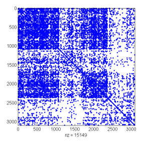
Figure 11.3: Spy-plots of 3 views in DBLP-1



(a) citation



(b) co-auth.



(c) co-term

Figure 11.4: Spy-plots of 3 views in DBLP-2

The second view connects two authors if they co-author at least one paper together. Finally, the third view connects two authors who share at least three terms in the title or abstract of their publications. Both datasets are portions of a larger dataset, manually extracted and labeled. In particular, DBLP-1 contains authors who published in venues STOC+FOCS, AAAI, SIGIR, TODS and DBLP-2 contains those published in venues ICDE, PODS, TKDE, CACM. That is, the ground truth clustering involves 4 author clusters for each of our real multi-networks. We remind the reader that we have analyzed the rank structure of DBLP-1 in Chapter 8.

In Fig.s 11.3 and 11.4 we illustrate the views for each one of our real datasets.

11.6.2 Clustering accuracy

In order to evaluate the performance of our proposed methods, we use the Normalized Mutual Information, a widely used metric for computing clustering accuracy of a method against the desired ground truth clustering [MRS08]. Moreover, we compare our methods, in terms of NMI, with two baseline approaches, which we briefly describe in the sequel:

BASELINE-1 algorithm sums all the adjacency matrices of a multi-graph obtaining a new aggregate sum-matrix and applies a k-way spectral clustering over this aggregate [vL07]. The k-way spectral clustering is based on the k-means algorithm that is applied on the Laplacian of the sum-matrix.

BASELINE-2 algorithm first constructs the spectral kernel for each graph view and then sums the spectral kernels summarizing all the dimensions of the multi-graph. Successively, the k-means algorithm is applied to the matrix containing the sum of the kernels in order to obtain the final clustering. Details for this algorithm may be found in [TLD09].

In Table 11.1 we show the NMI results on all datasets for all methods. We observe that MULTICLUS always outperforms baseline methods on all synthetic datasets. As for GRAPHFUSE, it has good performance over SYNTHETIC-1 and SYNT-2-SIM while, for SYNT-3-DIF, the results are on par with the baselines. Recall that by construction SYNT-3-DIF is difficult to cluster (see Fig.11.2 bottom), hence the drop in performance for all methods.

With respect to the real datasets, GRAPHFUSE obtains the best scores over both DBLP-1 and DBLP-2, while MULTICLUS has comparable behaviour with the baselines. We notice that NMI scores are overall lower on real datasets, as they have much less structure than the synthetic ones (see Fig.11.3) in addition to a lot more noise (see Fig.11.4). Nevertheless, GRAPHFUSE achieves significantly better accuracy compared to other methods. These encouraging results underline the merits of modeling the multi-graph clustering problem using tensors, as they seem to well exploit the interrelations of the views.

Dataset	BASELINE-1	BASELINE-2	MULTICLUS	GRAPHFUSE-1	GRAPHFUSE-2
SYNTHETIC-1	0.77 ± 0.11	0.96 ± 0.06	$\mathbf{1} \pm 0$	$\mathbf{1} \pm 0$	$\mathbf{1} \pm 0$
SYNT-2-SIM	0.68 ± 0.12	0.97 ± 0.11	$\mathbf{1} \pm 0$	$\mathbf{1} \pm 0$	$\mathbf{1} \pm 0$
SYNT-3-DIF	0.54 ± 0.01	0.56 ± 0.02	$\mathbf{0.90} \pm 0.01$	0.51 ± 0.17	0.67 ± 0.12
DBLP-1	0.12 ± 0.00	0.08 ± 0.01	0.11 ± 0.01	$\mathbf{0.30} \pm 0.02$	0.29 ± 0.02
DBLP-2	0.08 ± 0.01	0.04 ± 0.00	0.04 ± 0.00	$\mathbf{0.12} \pm 0.02$	0.09 ± 0.02

Table 11.1: **Proposed methods outperform the baselines**: NMI clustering accuracy of proposed methods and competitors on all datasets. Our proposed methods achieve superior performance for all clustering tasks.

11.6.3 Do more graph views help?

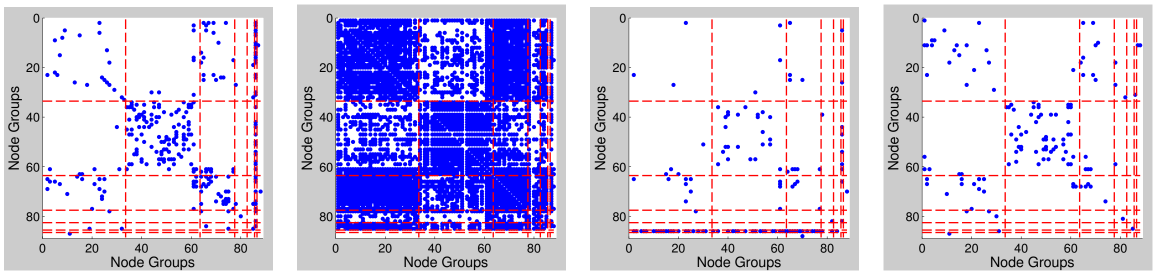
This question is one of the fundamental motivations of this work. Simply put, we want to understand whether the addition of more, different views of a given multi-graph is beneficial to the overall clustering quality. As a convention, we assume that all views are given a fixed number from 1 to K.

First, we want to evaluate if the mere presence of more views itself is beneficial, *on average*, for the clustering accuracy. In order to do that, we simply iterate over all possible combinations of $r = 1 \cdots K$ views of varying number and measure the NMI based on GRAPHFUSE. For DBLP-1 and DBLP-2, with only 2 views each, we measured average NMI respectively equal to 0.3037 and 0.0948, whereas adding a third view improved average accuracy to 0.3131 and 0.1208. Note that for this experiment, we report the maximum NMI of the two modes I and J of the tensor (although the trend is followed by both modes).

A second question we address is, how the clustering performance on a set of views R and another set of views C compare to each other, when $C \subset R$. Intuitively we would expect that the set R (the one with more views) allows us to obtain better results, in terms of NMI, than the set C. Our tests of the above hypothesis on DBLP-1 showed that NMI does *not* always increase monotonically with more views. For example, for view-1 and view-2 we obtained NMI=0.2844 where adding view-3 increased NMI to 0.3010. On the flipside, for a different ordering, we obtained NMI=0.3346 for two views and adding the third view caused the NMI to drop to 0.3009. Same behaviour was observed for DBLP-2. This demonstrates that while adding more views helps *on average*, adding a noisy view to a set of informative views might hurt the clustering accuracy for certain cases.

11.6.4 Data mining case study: **`Reality Mining`**

In this section, we provide a data mining case study on the `Reality Mining` dataset. We remind the reader that we analyze the rank structure of `Reality Mining` in Chapter 8. This dataset was introduced in [EPL09] and contains data collected by the MIT Media Lab, including subjects (undergraduate and graduate CS and business students) whose interactions were monitored by a pre-installed piece of software on their mobile devices. The different views offered by the dataset pertain to the means of interaction between a pair of subjects. Namely, `CALL` view refers to subjects calling each other, `DEVICE` view contains Bluetooth device scans, `SMS` view is constructed based on text message exchanges, and `FRIEND` view contains friendship claims.



(a) `CALL` (b) `DEVICE` (c) `SMS` (d) `FRIEND`

Figure 11.5: **GRAPHFUSE outputs communities that agree with ground truth**: Results on the four views of the `Reality Mining` multi-graph. Red dashed lines outline the clustering found by GRAPHFUSE.

In Fig.11.5, we show all four views of the dataset as clustered by GRAPHFUSE where $R = 6$. Qualitatively, we see that the algorithm's output agrees with the communities that appear to be strong on the spy-plots of each view. For example, cluster 2 is a community of business school students that are mostly isolated from the rest of the graph. Another example is cluster 6 of size 1, which contains a single subject with many incoming calls and many outgoing SMSs.

11.7 Conclusions

In this chapter we address the multi-graph clustering problem, where the goal is to find well-defined clusters across *all* the views (a.k.a. dimensions, layers) of a given graph. We propose two different solutions for clustering multi-graphs, based on Minimum Description Length and Tensor-based decomposition principles, respectively. We validate the effectiveness of our techniques over both synthetic and real `DBLP` networks, obtaining better clustering accuracy than two competitor methods that ignore the multi-view aspect of the networks. Our case study on the real `Reality Mining` data reveals interesting clusters that agree with human intuition. Moreover, we show that our tensor-based method is a generalization of a recent approach [TLD09], under appropriate conditions. Finally, we asses how the clustering process benefits from the existence of multiple views. In short, the presence of more views is beneficial *on average*, but for particular instances, addition of noisy views may deteriorate clustering quality. This outcome paves the way for interesting research questions, e.g. how to select only informative and non-redundant views of the multi-graph or how to weigh the different dimensions appropriately to obtain the best clustering accuracy.

Chapter based on material published in [AGP15a, AGP15b].

Chapter 12

Homogeneity in Web Search Results: Diagnosis and Mitigation

Comparing semantically the results of different search engines and using social media signals to diversify web search.

In this Chapter we propose two novel tools for comparing semantically the results of different search engines. We present our empirical study over the search results produced by Google and Bing that shows a large overlap. Fortunately, our study also shows that by mining Twitter data one can obtain search results that are quite distinct from those produced by Google and Bing. Additionally, through user studies, we show that users found those results to be quite informative, presenting an potential alternative to traditional web search.

12.1 Introduction

The fairness doctrine contends that citizens should have access to diverse perspectives as exposure to different views is beneficial for the advancement of humanity [Fed49]. The World Wide Web is now widely recognized as the universal information source. Content representing diverse perspectives exist on the Web, on almost on any topic. However, this does not automatically ensure that citizens encounter them [SM12b].

Search engines have become the primary tool used to access the web content [PBR12]. In particular, it is the duopoly of Google and Bing that largely arbitrates which documents people see, especially from the English language web (Yahoo's web search is currently powered by Bing). In the physical world, people gain access to diverse perspectives by subscribing to different newspapers, listening to different radio stations, tuning into different television channels, or manually selecting different publications and books. We seek to study whether users can gain different perspectives by obtaining results for the same query from different search engines.

In addition to the information about the documents that the search engine deems most relevant to the query (the so called "organic results"), a search engine result page (SERP) often contains a variety of other information. This may include inter alia sponsored listings, images, videos, maps, definitions, or suggested search refinements. We focus on comparing the top-10 organic results on the first SERP because they are the ones that get most of the clicks [ESSF12]. Users unsatisfied with the top results frequently retype a query, instead of looking at results at lower positions [GC07]. An organic result normally includes the title of the document, a snippet of the document, and URL of the full version. Thus, the first research question this Chapter investigates is how distinctive are the organic web search results provided by Google and Bing.

In parallel, social networks have become immensely popular and have come to dominate the zeitgeist of modern life. The second research question this Chapter investigates is whether data mining of social networks can help web search engines diversify their search results [CCSV11, MGD+09]. Not only the social results must be distinct from the web results for the same query, the users must also find them useful. We use Twitter data in this exploration because it is still possible to selectively crawl Twitter.

Contributions In this work, our main contribution lies in quantifying how distinctive are the organic search results produced by Google, Bing, and Twitter. In order to achieve that, we also make the following technical contributions:

- **Visualization and exploratory analysis:** We introduce TENSORCOMPARE, an exploratory tool for visualizing and analyzing pairwise differences between search engines.
- **Quantitative comparison of search engines results**: We also introduce CROSSLEARN-COMPARE, a tool that uses machine learning and quantifies the similarity of results between two search engines by framing it as a prediction problem.
- **Method Generality**: While designed to effectively analyze search engine results, our tools have broader applicability. For instance, consider a set of questions, possibly coming from a Massive Open Online Course (MOOC) exam; these questions can either be multiple choice or in free-text form. In the setting of a MOOC, there will be potentially hundreds, or thousands, of students responding to those questions. There are also multiple exams, as well as multiple MOOCs on the same subject, offered by different providers. Using these tools, we are able to quantify the similarity of students across different MOOCs, as well as similarity of MOOCs in terms of how students respond to exam questions (which could be an indicator of how well students learn from a particular MOOC).

The Chapter is organized as follows. We begin by discussing related work in Section 12.2. We then describe the new tools we designed in Section 12.3. Section 12.4 presents the setup for the empirical evaluation. Section 12.5 presents the quantification of the overlap between Google and Bing web results, while Section 12.6 presents this quantification for their results and that of a simple search engine built over Twitter tweets. Section 12.7 presents the user study for assessing the usefulness of our findings. We conclude with a discussion of the significance of the work and future directions in Section 10.7.

12.2 Related Work

More than four decades ago, Lancaster and Fayen [LF73] in 1973 listed six criteria for assessing the performance of information retrieval systems: 1) Coverage, 2) Recall, 3) Precision, 4) Response time, 5) User effort, and 6) Form of output. Since the advent of search engines in early 90's, there are several reported studies that evaluated their performance on one or more these criteria. See [CR96] and references therein for examples of some early studies. See [Lew12] for a recent compilation of various issues and studies related to the evaluation of web search engines. We will focus our discussion on prior works that studied the overlap of results between different search engines, the thrust of our paper.

Three lines of research are most relevant to our work: i) overlap between the results of web search engines, ii) social search technologies, and iii) integration of social search results into web search. We review all three in this section. Note that we use the term "social search" to mean searches conducted over databases of socially generated content, although this term often refers broadly to the process of finding information online with the assistance of any number of social resources such as asking others for answers or two people searching together [TRM11].

12.2.1 Overlap in Web Search Results

An early overlap study is due to Ding and Marchionini, who measured the result overlap between the then popular three search engines: InfoSeek, Lycos, and OpenText. Five queries were used to conduct searches with these services. They observed a low level of result overlap among the services [DM96]. Around the same time, Selberg and Etzioni found that each of Galaxy, Infoseek, Lycos, OpenText, Webcrawler and Yahoo returned mostly unique results [SE95]. Also in 1996, Gauch, Wang and Gomez found that a metasearch engine that fused the results of Alta Vista, Excite, InfoSeek, Lycos, Open Text, and WebCrawler provided the highest number of relevant results [GW96]. Bharat and Broder estimated the overlap between the Websites indexed by HotBot, Alta Vista, Excite and InfoSeek in November 1997 to be only 1.4% [BB98]. Lawrence and Giles, in their study of AltaVista, Excite, HotBot, Infoseek, Lycos, and Northern Light published in 1998, found that the individual engines covered from 3 to 34% of the indexable Web [LG98]. Spink et al. studied the overlap between the results of four search engines, namely MSN (predecessor of Bing), Google, Yahoo and Ask Jeeves, using data from July 2005. They found that the percent of total first page results unique to only one of the engines was

84.9%, shared by two of the three was 11.4%, shared by three was 2.6%, and shared by all four was 1.1% [SJBK06]. In an update two years later, they noted that the first page results of the four engines continued to differ from one another and in fact they included fewer results in common in 2007 than in 2005 [SJW08].

More recently, Pirkola investigated how effectively the websites of Finnish, French, and U.S. domains were being indexed by two US-based and three Europe-based search engines [Pir09]. The results showed that Google and Live Search (predecessor of Bing) indexed US sites more effectively than Finnish and French sites, the Finnish www.fi indexed only Finnish sites and the French Voila only French sites, and the European engine Virgilio indexed European sites more effectively than US sites. In another interesting study, Wilkinson and Thelwall compared the results of seventeen random queries submitted to Bing for thirteen different English geographic search markets at monthly intervals [WT13]. They found there were almost no ubiquitous authoritative results: only one URL was always returned in the top-10 for all search markets and points in time and that results from at least three markets needed to be combined to give comprehensive results. There also have been studies pointing out that the search engine results are not stable even in short windows of time [BI04, Lew12].

We did not find much discussion in prior work of the techniques used for determining if two result pages contained links to the same web document. For example, [SJBK06, SJW08] simply state that this determination is done using string comparison of URLs. It is not clear what URL normalization [LKH05, LCY+10], if any, was done before string comparison. It is also not clear what, if anything, was done to address the problem of DUST - Different URLs with Similar Text [BYKS09]. Finally, there is no mention of short URLs, although the first notable URL shortening service, namely tinyURL, dates back to 2002 [APK+11].

To summarize, all of prior work found little overlap between the first page results produced by different web search engines for very many queries. Some plausible reasons have also been put forward for this low overlap. They include that the search engines are constrained in the portions of the Web they index due to network bandwidth, disk storage, computational power, or a combination of these items. Search engines use different technologies to find pages and indexing them. And they deploy proprietary algorithms to determine the ranking of the results and their presentation to the users. Fingers have also been pointed at implicit personalization [HSMK+13].

One way the users dealt with low overlap was by manually executing the same query on multiple search engines. Analyzing six months of interaction logs from 2008-2009, White and Dumais [WD09] found that 72.6% of all users used more than one engine during this period, 50% switched engines within a search session at least once, and 67.6% used different engines for different sessions. Their survey revealed three classes of reasons for this behavior: dissatisfaction with the quality of results in the original engine (dissatisfaction, frustration, expected better results, totaling 57%), the desire to verify or find additional information (coverage/verification, curiosity, totaling 26%), and user preferences (destination preferred, destination typically better, totaling 12%). Another way the problem of low overlap was addressed was by developing metasearch engines (e.g. InFind, MetaCrawler, MetaFerret, ProFusion, SavvySearch). A metasearch engine automatically queries a number of search engines, merges the returned lists of results, and presents the resulting ranked list to the user as the search of the query [AM01, GS05, MYL02]. Note that with either manual or automated approach, the user ends up seeing multiple perspectives.

In sharp contrast with prior work, our study conducted using data from June-July 2014 and presented here, finds large overlap between the top-10 search results produced by Google and Bing. This overlap is even more pronounced in the top-5 results and the results of head queries. Some plausible reasons for greater convergence in the search results include deployment of greater amount of resources by search engines to cover a larger fraction of indexable Web, much more universal understanding of search engine technologies, and the use of similar features in ranking the search results. A consequence of this convergence is that it becomes now harder for people to access diverse perspectives.

12.2.2 Social Search

In addition to being considered a social media and a social network [KLPM10], Twitter may also be viewed as an information retrieval system that people can utilize to produce and consume information. Twitter today receives more than 500 million tweets per day at the rate of more than 33,000 tweets per second. More than 300 billion tweets have been sent since the founding of Twitter in 2006 and it receives more than 2 billion search queries every day. Twitter serves these queries using an inverted index tuned for real-time search, called EarlyBird, described in [BGL+12]. While this search service excels at surfacing breaking news and events in real time and it does indeed incorporate relevance ranking, it is a feature that the system designers themselves consider that they have "only begun to explore".[1]

The prevailing perception is that much of the content found on Twitter is of low quality [ACG+10] and the keyword search as provided by Twitter is not effective [TAHH12]. In response, there has been considerable research aimed at designing mechanisms for finding good content from Twitter. In many of the proposed approaches, retweet count alone or in conjunction with textual data, author's metadata, and propagation information play a prominent role [CMP11, DJQ+10, TAHH12, Web14]. The intuition is that if a tweet is retweeted multiple times, then several people have taken the time to read it, decide it is worth sharing, and then actually retweeted it, and hence it must be of good quality [UC11]. But, of course, one needs to remove socware and other spam before using retweet count [MRA13, RHMF12, SMML+14] Other approaches include using the presence of a URL as an indicator [ACG+10], link analysis on the follows and retweet

[1]One of the problems with Twitter search has been that, while it is easy to discover current tweets and trending topics, it is much more difficult to search over older tweets and determine, say, what the fans were saying about the Seahawks during the 2014 Super Bowl. Beginning November 18, 2014, however, it has become possible to search over the entire corpus of public tweets. Still, our own experiments indicate that the ranking continues to be heavily biased towards recency.

graphs [RGAH11, YLLR12], clustering taking into account the size and popularity of a tweet, its audience size, and recency [LNK10], and the semantic approaches including topic modeling [YR14]. See overviews in [Web14, YR14] for additional references.

In this work, we are not striving to create the best possible social search engine, but rather investigate whether the results obtained using signals from a social network could be substantially different from a web search engine and yet useful. Thus, in order to avoid confounding between multiple factors, we shall use a simple social search engine that ranks tweets based on retweet analysis.

Contrary to the rich literature on overlap between the results produced by the web search engines, the only prior work we could find on overlap between web and social search results appears in Section 5 of [TRM11] (TRM Study). They extracted snippets of all search results from Bing search logs for 42 most popular queries for one week in November 2009. They also obtained all the tweets containing those queries during the same period. They then computed per query average cosine similarity of each web snippet with the centroid of the other web snippets and with the centroid of the tweets. Similarly, they computed the per-query average cosine similarity of each Twitter result with the centroid of the other tweets and with the centroid of the web snippets. All averaging and comparisons are done in the reduced topic space obtained using Latent Dirichlet Allocation (LDA) [BNJ03]. They found that the average similarity of Twitter posts to the Twitter centroid was higher than the web results' similarity to the web centroid. The issue of usefulness of Twitter results is not addressed in their paper.

We shall see that our study considers head as well as trunk queries and encompasses both Google and Bing. We also employ different data mining tools in our study. Specifically, our TensorCompare uses tensor analysis to obtain low-dimensional representation of search results since the method of moments for LDA reduces to canonical decomposition of a tensor, for which scalable distributed algorithms exist [AGH+14, KPHF12]. Our CrossLearnCompare, uses a novel cross-engine learning to quantify the similarity of snippets and tweets. Additionally, we provide a user study demonstrating the usefulness of the Twitter results. We will have more to say quantitatively about the TRM study when we present our experimental results.

12.2.3 Integration of Web and Social search

Bing has been including a few tweets related to the current query on its search result page, at least since November 2013. However, it is not obvious for what queries this feature is triggered and what tweets are included. For example, on February 12, 2015 at 10:42AM, our query "Greece ECB" brought only one tweet on Bing's result page, which was a retweet from Mark Rauffalo from two days ago. Bing also offered a link titled "See more on Twitter" below this tweet. Clicking this link took us to a Twitter page, where the top tweet was from 14 minutes ago with the text "ECB raises pressure on Greece as Tsipras meets EU peers"! Since June 2014, one can also search Bing by hashtag, look up specific Twitter handles, or search for tweets related to a specific celebrity. Google is also said have struck a deal with Twitter that will allow tweets to be shown in Google search results sometime during 2015.

There is also research on how web search can be improved using signals from Twitter. For example, Rowlands et al. [RHS10] propose that the text around a URL that appears in a tweet may serve to add supplementary terms or add weight to existing terms in the corresponding web page and that the reputation or authority of the tweeterer may serve to weight both annotations and query independent popularity. Similarly, Dong et al. [DZK$^+$10] advocate using Twitter stream for detecting fresh URLs as well as for computing features to rank them. We propose to build our future work upon some of these ideas.

12.3 Analytical Tools

We designed two tools to be able to analyze and compare search engine results. One, which we call TensorCompare, uses tensor analysis to derive low-dimensional compact representation of search results and study their behavior over time. The other, which we call CrossLearnCompare, uses cross-engine learning to quantify their similarity. We discuss them next.

12.3.1 TensorCompare

Postulate that we have the search results of executing a fixed set of queries at certain fixed time intervals on the same set of search engines. These results can be represented in a four mode tensor $\underline{\mathbf{X}}$, where (query, result, time, search engine) are the four modes [KB09]. A result might be in the form of a set of URLs or a set of keywords representing the corresponding pages. The tensor might be binary valued or real valued (indicating, for instance, frequencies).

This tensor can be analyzed using the so-called canonical or PARAFAC decomposition [Har70, Bro97], which decomposes the tensor into a sum of rank-one tensors: $\underline{\mathbf{X}} \approx \sum_{r=1}^{R} \lambda_r \; \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r \circ \mathbf{d}_r$, where the (i, j, k, l)-th element of $\mathbf{a} \circ \mathbf{b} \circ \mathbf{c} \circ \mathbf{d}$ is simply $\mathbf{a}(i)\mathbf{b}(j)\mathbf{c}(k)\mathbf{d}(l)$. The vectors $\mathbf{a}_r, \mathbf{b}_r, \mathbf{c}_r$, $\mathbf{d}_r$ are usually normalized, with their scaling absorbed in λ_r. For compactness, the decomposition is represented as matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$.

The decomposition of $\underline{\mathbf{X}}$ to $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ gives us a low rank embedding of queries, results, timings, and search engines respectively, corresponding to the aforementioned clusters. This implies that we are able to track the temporal behavior of a cluster of semantically similar search engines for a set of queries. The factor matrix $\mathbf{D}$ projects each one of the search engines to the R-dimensional space. Alternatively, one can view this embedding as soft clustering of the search engines, with matrix $\mathbf{D}$ being the cluster indicator matrix: the (i, j) entry of $\mathbf{D}$ shows the participation of search engine i in cluster j.

This leads to a powerful visualization tool that captures similarities and differences between the search engines in an intuitive way. Say we take search engines A and B and the corresponding rows of matrix $\mathbf{D}$. If we plot these two row vectors against each other, the resulting plot will contain as many points as clusters (R in our particular notation).

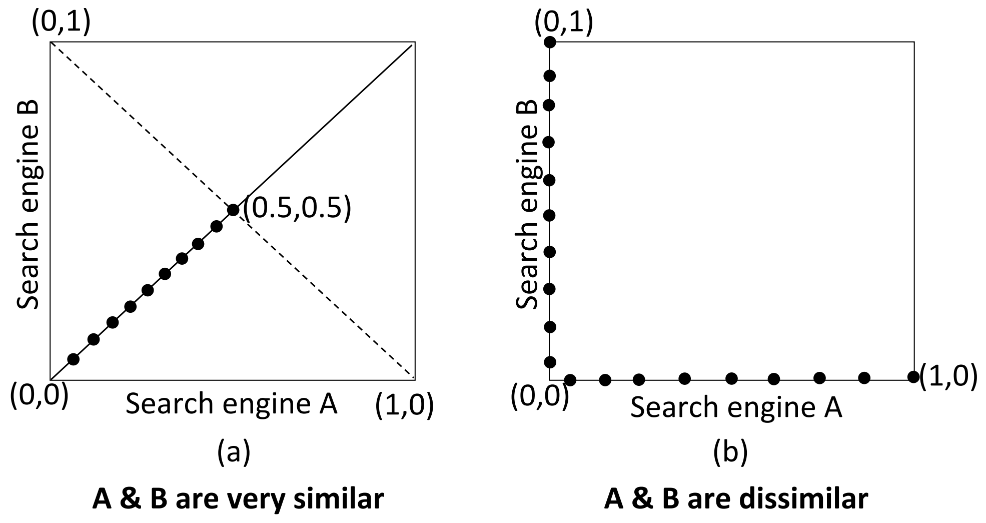


Figure 12.1: Visualization guide for TENSORCOMPARE.

The positions of these points are the key to understanding the similarity between search engines.

Figure 12.1 serves as a guide. The (x, y) coordinate of a point on the plot corresponds to the degree of participation of search engines A and B respectively in that cluster. If all points lie on the 45 degree line, this means that both A and B participate equally in all clusters. In other words, they tend to cluster in the exact same way for semantically similar results and for specific periods of time. Therefore, Fig. 12.1(a) paints the picture of two search engines that are very (if not perfectly) similar with respect to their responses. In the case where we have only two search engines, perfect alignment of their results in a cluster would be the point $(0.5, 0.5)$. If we are comparing more than two search engines, then we may have points on the lower parts of the diagonal. In the figure, we show multiple points along the diagonal for the sake of generality.

Figure 12.1(b), on the other hand, shows the opposite behavior. Whenever a point lies on either axis, this means that only one of the search engines participate in that cluster. If we see a plot similar to this figure, we can infer that A and B are very dissimilar with respect to their responses. In the case of two search engines, the only valid points on either axis are $(0, 1)$ and $(1, 0)$, indicating an exclusive set of results. However, for generality, we show multiple points on each axis.

Note, of course, the cases shown in Fig. 12.1 are the two extremes, and we expect to observe behaviors bounded by those extremes. For instance, in the case of two search engines, all points should lie on the line $\mathbf{D}(1, j)x + \mathbf{D}(2, j)y = 1$, where $\mathbf{D}(1, j)$ is the membership of engine A in cluster j, and $\mathbf{D}(2, j)$ is the membership of engine B in cluster j. This line is the dashed line of Fig. 12.1(a).

TENSORCOMPARE also allows us to track the behavior of clusters over time. In particular, given the i-th group of semantically similar (query, result, search engine) cluster, as given by the decomposition, the i-th column of matrix **C** holds the temporal profile of that cluster. Suppose we have T days worth of measurements. If the search engines of that cluster produce similar results for the given set of queries for all T, the temporal profile will be approximately constant and each value will be approximately equal to $\frac{1}{T}$. Otherwise, there will be variation in the profile, correlated with the variation of the particular results. In the extreme case where a result appeared only on a single day, the time profile will have the value approximately equal to one corresponding to that day, and approximately zero for the rest of the days.

Theoretical Foundation We next provide a Lemma that connects the plots provided by TENSORCOMPARE to the degree of semantic overlap of two search engines. Suppose that for a given cluster j, we denote the membership of search engine A as $x = D(A, j)$ and the membership of search engine B as $y = D(B, j)$. For ease of exposition, consider the case of two search engines and assume that we have a three mode tensor: (query, result, search engine).

Lemma 12.1.:
Assume a binary (query, result, search engine) tensor that has exactly one rank one component. Let search engine A correspond to the x coordinate, and search engine B correspond to the y coordinate of a TENSORCOMPARE plot. For the particular component, if search engine B has p_1 fraction of queries in common with A, and p_2 portion of the result in common with A, then

$$y \leq p_1 p_2 x.$$

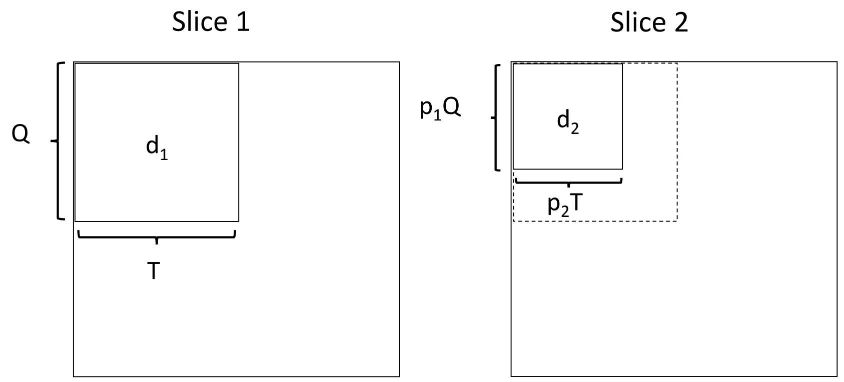


Figure 12.2: The two slices of $\underline{\mathbf{X}}$.

Proof. Consider a tensor $\underline{\mathbf{X}}$ with dimensions $I \times J \times 2$ (in our case, the first mode corresponds to queries, the second to results, and the third to search engines). Assume that $\underline{\mathbf{X}}$ is rank one, which means that there is one component in its PARAFAC decomposition. In the frontal slice corresponding to the first search engine (Slice 1 in Fig. 12.2), we have Q queries and T results forming a perfect block, which we assume to be filled with 1's. The second slice, which corresponds to the second search engine, has a block that spans only a fraction of the queries and results of Slice 1.

Algorithm 12.1: CROSSLEARNCOMPARE

```
Input: $\mathcal{R}_A$, $\mathcal{R}_B$ are instances of results of engines A and B. Each instance is in the form (query,
    result representation in chosen feature space)
Output: Similarity measures $c_{A,B}$ and $c_{B,A}$ between search engines A, B.
1: Train a model $\mathcal{M}_A$ based on the instances $\mathcal{R}_A$, using the query as a class label.
2: Train a model $\mathcal{M}_B$ based on the instances $\mathcal{R}_B$, using the query as a class label.
3: For all instances in $\mathcal{R}_B$, use $\mathbf{M}_A$ to predict the query. Set $c_{A,B}$ as a measure of the classifier's
    accuracy (e.g. Area Under the Curve).
4: For all instances in $\mathcal{R}_A$, use $\mathbf{M}_B$ to predict the query. Set $c_{B,A}$ likewise.
```

We assume that the components $\mathbf{a}, \mathbf{b}$ of the PARAFAC decomposition are normalized by their ℓ_2 norm, and the scaling is absorbed in $\mathbf{c}$. We further assume that the components are non-negative.

Let $\hat{\mathbf{a}}$, $\hat{\mathbf{b}}$, $\hat{\mathbf{c}}$ be the optimal solution. An upper bound $\mathbf{a}, \mathbf{b}, \mathbf{c}$ to the optimal is the following: The first Q elements of $\mathbf{a}$ will be equal to $\frac{1}{\sqrt{Q}}$ (the rest are zero), and the first T elements of $\mathbf{b}$ will equal $\frac{1}{\sqrt{T}}$. This implies that the coefficients of $\mathbf{c} = \begin{bmatrix} c_1 & c_2 \end{bmatrix}^2$, which multiply $\mathbf{a}\mathbf{b}^T$ in order to approximate the respective slices of $\underline{\mathbf{X}}$, will be proportional to the respective densities of the blocks in either slice, i.e. $c_1 \propto d_1$ and $c_2 \propto d_2$ Making this uniformity assumption for the non-zero elements of $\mathbf{a}, \mathbf{b}$ allows us to bound the ratio of the coefficients of $\hat{\mathbf{c}}$ by the ratio of the densities of the blocks in each slice. More specifically, we have

$$\frac{\hat{c}_1}{\hat{c}_2} \leq \frac{d_1}{d_2} = \frac{QT}{p_1 Q p_2 T} = \frac{1}{p_1 p_2}.$$

Hence, $\hat{c}_2 \leq p_1 p_2 \hat{c}_1$. If we substitute $y = \hat{c}_2$ and $x = \hat{c}_1$, as they correspond in Fig. 12.1, then we have shown the desired upper bound. ■

In the case of a four-mode tensor, with p_3 percent overlap in the time mode, the bound is $y \leq p_1 p_2 p_3 x$. The above Lemma provides an upper bound, however, we experimentally validated that this bound is in practice tight.

12.3.2 CrossLearnCompare

An intuitive measure of the similarity of the results of two search engines is the predictability of the results of a search engine given the results of the other. Say we view each query as a class label. We can then go ahead and learn a classifier that maps the search result of search engine A to its class label, i.e. the query that produced the result. Imagine now that we have results that were produced by search engine B. If A and B return completely different results, then we would expect that classifying correctly a result of B using the classifier learned using A's results would be difficult, and our classifier would probably err. On the other hand, if A and B returned almost identical results, classifying correctly the search results of B would be easy. In cases in between, where A and B bear some level of similarity, we would expect our classifier to perform in a way that it is correlated with the degree of similarity between A and B.

Note we can have different accuracy when predicting search engine A using a model trained on B, and vice versa. This, for instance, can be the case when the results of A are a superset of the results of B. Algorithm 12.1 shows an outline of CROSSLEARNCOMPARE.

12.4 Experimental Setup

We next describe the experimental setup of the empirical study we performed, applying the tools just described.

12.4.1 Social Pulse

For concreteness, we first specify a simple social search engine, which we shall henceforth refer to as Social Pulse. We are not striving to create the best possible search engine, but rather investigate whether the results obtained using signals from a social network could be substantially different from a Web search engine and yet useful. Thus, instead of employing a large set of features (see Section 12.2.2), we purposefully base the Social Pulse's ranker on one single feature in order to be able to make sharp conclusions and to avoid confounding between multiple factors.

Social Pulse uses Twitter as the social medium. For a given query, Social Pulse first retrieves all tweets that pertain to that query. Multiple techniques are available in the literature for this purpose (e.g. [BKML13, NHF12, SFD+10, TT12]). We choose to employ the simple technique of checking for the presence of the query string in the tweet. Subsequently, Social Pulse ranks the retrieved tweets with respect to the number of re-tweets (more precisely, the number of occurrences of the exact same tweet without having necessarily been formally re-tweeted).

Arguably, one can restrict the attention to only those tweets that contain at least one URL [ACG+10]. However, we have empirically observed that highly re-tweeted tweets, in spite of containing no URL, usually provide high quality result. Hence, Social Pulse uses these tweets as well.

12.4.2 Data Set

We conducted the study for two sets of queries. The TRENDS set (Table 12.1) contains the most popular search terms from different categories from Google Trends during April 2014. We will refer to them as *head queries*. The MANUAL set (Table 12.2) consists of hand-picked queries by the authors that we will refer to as *trunk queries*. These queries consist of topics that the authors were familiar with and were following at the time. Familiarity with the queries is helpful in understanding whether two sets of results are different and useful. Queries in both the sets primarily have the informational intent [Bro02]. Many of them are named entities, which constitute a significant portion of what people search. The total number of queries was limited by the budget available for the study.

We probed the search engines with the same set of queries at the same time of the day for a period 21 days for the TRENDS set, and 17 days for the MANUAL set, during June-July

Albert Einstein	American Idol	Antibiotics	Ariana Grande
Avicii	Barack Obama	Beyonce	Cristiano Ronaldo
Derek Jeter	Donald Sterling	Floyd Mayweather	Ford Mustang
Frozen	Game of Thrones	Harvard University	Honda
Jay-Z	LeBron James	Lego	Los Angeles Clippers
Martini	Maya Angelou	Miami Heat	Miami Heat
Miley Cyrus	New York City	New York Yankees	Oprah Winfrey
San Antonio Spurs	Skrillex	SpongeBob SquarePants	Tottenham Hotspur F.C.
US Senate			

Table 12.1: TRENDS queries

Afghanistan	Alternative energy	Athens	Beatles	Beer
Coup	Debt	Disaster	E-cigarettes	Education
Gay marriage	Globalization	Gun control	IMF	iPhone
Iran	Lumia	Malaria	Merkel	Modi
Paris	Polio	Poverty	Rome	Russia
San Francisco	Self-driving car	Syria	Tesla	Ukraine
Veteran affairs	World bank	World cup	Xi Jinping	Yosemite

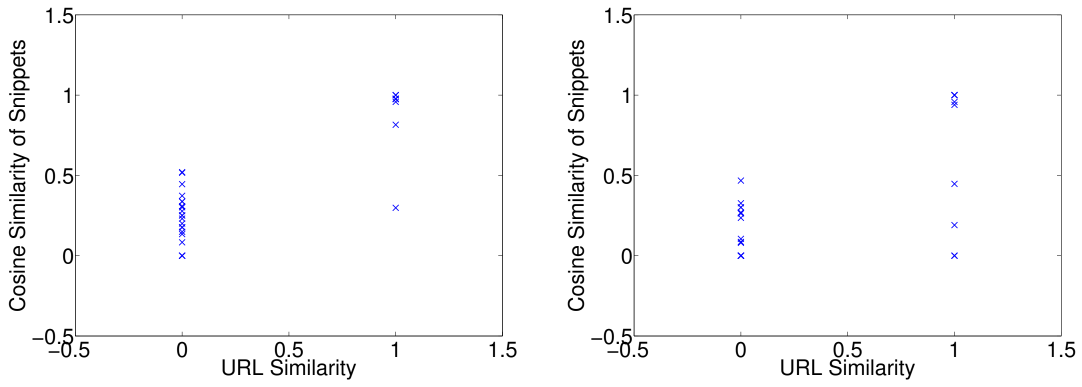
Table 12.2: MANUAL queries

2014. For Google, we used their *custom search API* (code.google.com/apis/console), and for Bing their *search API* (datamarket.azure.com/dataset/bing/ search). Twitter data consists of 1% sample of tweets obtained using Twitter API.

In all cases, we recorded the top-k results. The value of k is set to 10 by default, except in the experiments studying the sensitivity of results to the value of k. Every time, we ran the same code from the same machine having the same IP address to minimize noise in the results. Because we were getting the results programmatically through the API, no cookies were used and there was no browser information used by Google or Bing in producing the results [HSMK+13].

12.4.3 Representation of Search Results

While our methodology is independent of the specific representation of search results, we employ the snippets of the search results provided by the search engines for this purpose. The snippet of a search result embodies the search engine's semantic understanding of the corresponding document with respect to the given query. The users also heavily weigh the snippet in deciding whether to click on a search result [MGC10]. The alternative of using URL representation must first address the well-known problems arising from short URLs [APK+11], un-nomalized URLs [LKH05, LCY+10], and different URLs with similar text [BYKS09]. Unfortunately, there is no agreed upon way to address them and the specific algorithms deployed can have large impact on the conclusions. Furthermore, the users rarely decide whether to look at a document based on the URL they see on the search result page [MGC10]. In the case of Social Pulse, the entire text of a tweet



(a) TRENDS query set (b) MANUAL query set

Figure 12.3: **Comparing URL similarity with snippet similarity**: snippet similarity is a good proxy for real URL similariy, as judged by a human who manually visited each URL.

(including hashtags and URLs, if any) is treated as snippet for this purpose. Snippets and tweet texts respectively have also been used in the study of overlap between the results of web search and social search in [TRM11].

More in detail, for a given result of a particular query, on a given date, we take the bag-of-words representation of the snippet, after eliminating stopwords. Subsequently, a set of results from a particular search engine, for a given query, is simply the union of the respective bag-of-words representations. For TENSORCOMPARE, we keep all words and their frequencies; binary features did not change the trends. For CROSSLEARNCOMPARE, we keep the top-n words and have binary features. Finally, we note that the distribution of the snippet lengths for Google, Bing, and Social Pulse was almost identical for all the queries we tested. This ensures a fair comparison between them.

To assess whether snippets are appropriate for comparing the search results, we conducted the following experiment. We inspect the top result given by Google and Bing for a single day, for each of the queries in both TRENDS and MANUAL datasets. If for a query, the top result points to the same content, we assign the URL similarity score of 1 to this query, and the score of 0 otherwise. We then compute the cosine similarity between the bag-of-word representations of the snippets produced by the two search engines for the same query. Figure 12.3 shows the outcome of this experiment. Each point in this figure corresponds to one query and plots the URL and snippet similarity scores for this query. For clarity, the X and Y axes show ranges beyond [0,1].

We see that for most of the queries for which the snippet similarity is low, the results point to different documents. On the other hand, when this similarity is high, the documents are identical. In both TRENDS and MANUAL, there exist some outliers with pointers to identical documents yet dissimilar snippets. Yet, overall, Fig. 12.3 indicates that snippets are good vehicles for content comparison.

Note that we do not consider their ordering in our representation of the search results. Instead, we study the sensitivity of our conclusions to the number of top results, including top-1, top-3, and top-5 (in addition to top-10).

12.5 Overlap Between Web Results

We first study the overlap between the web results by Google and Bing.

12.5.1 Results of TensorCompare

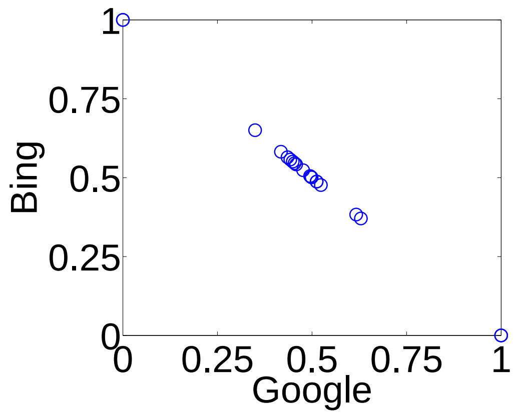
Having chosen a bag-of-words representation for the results, the input tensor to TENSORCOMPARE has modes (query, term, date, search engine). Our data collection results in a $32 \times 36631 \times 21 \times 2$ tensor for the TRENDS dataset and a $35 \times 39725 \times 17 \times 2$ tensor for the MANUAL set. For fitting the PARAFAC decomposition, we use the algorithm from [CK12] that is appropriate for sparse, count data. More specifically, we use Tensor Toolbox from Matlab [BK+15], which contains an efficient implementation of this algorithm. The number of components we chose was $R = 20$; however, qualitatively similar behavior was observed for various values for R. The results of TENSORCOMPARE analysis are shown in Figs. 12.4 and 12.5. Figure 12.4 shows the similarity of search results, while Fig. 12.5 shows the temporal profile of each one of the points in Fig. 12.4.

The first, immediate, observation is that the latent clusters for both query sets behave similarly. This fact is encouraging because it shows that our analysis can be applied to both head and trunk queries. In order to interpret the aforementioned plots, we consult Fig. 12.1. We observe that Google and Bing produce similar results. This is indicated by the fact that in Fig. 12.4, the majority of the points lie around the $(0.5, 0.5)$ point (we remind the reader that this point indicates almost exact similarity for the case of two search engines), showing near equal participation of Google and Bing to the majority of the latent clusters. This finding is quite surprising and is in sharp contrast with the past studies. We further observe that there are somewhat more results unique to Google than Bing since there are more clusters where Google has single participation.

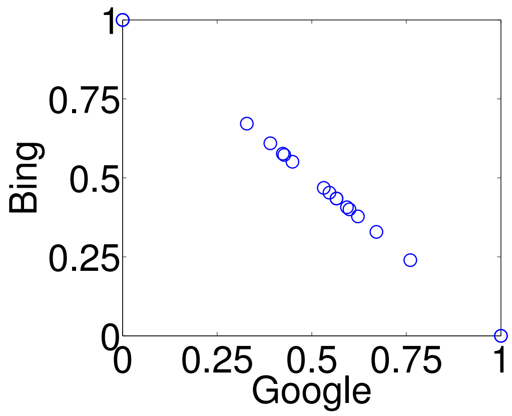
Finally, with respect to the temporal variation of the results, as indicated by Fig. 12.5, the temporal profile of each cluster is almost uniform across time. This, consequently, means that for both search engines, either in cases where they agree or in cases where they produce somewhat distinct results, their behavior is stable over time.

12.5.2 Results of CrossLearnCompare

We next present our analysis of the application of CROSSLEARNCOMPARE to the search results of two engines. To obtain feature space for our instances, we remove terms that are verbatim equal to or contain the query string and then take the 100 highest frequency words for each search engine. We use the union of these two bags of words as the feature space of the training and testing instances. Each such instance is, thus, the vector space representation of a result for a given date and position in the result-set. We use a binary representation, where 1 indicates that the corresponding word appears in the particular



(a) TRENDS query set



(b) MANUAL query set

Figure 12.4: **Visualization of TENSORCOMPARE for Google and Bing for top-10 results**: Values on the x-axis correspond to the membership of Google to a cluster, and values on the y-axis correspond to the membership of Bing. Thus, an (x, y) point on this plot represents one of the clusters of TENSORCOMPARE. The closer the points are to the 45-degree line, the more similar are the two search engines.

instance.

We train one-vs-all linear SVM classifiers for each query set, for each search engine. The performance of the two classifiers of CROSSLEARNCOMPARE for the two query sets is shown in Fig. 12.6; the measure of performance used is the standard Receiver Operating Characteristic (ROC) curve [BD06]. There are four curves on the same figure, showing the performance of predicting Bing using Google and vice versa, and for the two query sets. Table 12.3 includes the Area Under the Curve (AUC) for the ROC curves shown in Fig. 12.6.

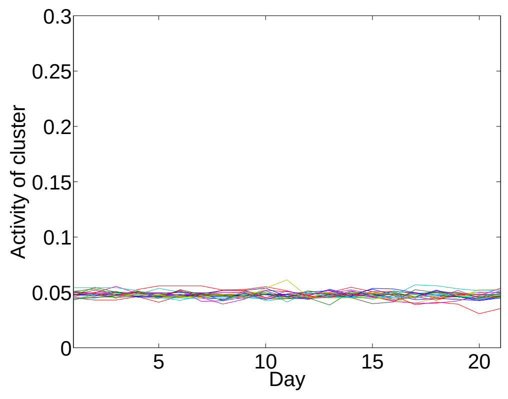
Google- Bing	TRENDS $\rightarrow$	TRENDS $\leftarrow$	MANUAL $\rightarrow$	MANUAL $\leftarrow$
top-10	1.0	1.0	0.92	0.73
top-5	0.81	1.0	1.0	0.78
top-3	0.80	1.0	1.0	0.77
top-1	0.97	1.0	1.0	0.22

Table 12.3: Area Under the Curve (AUC) for CROSSLEARNCOMPARE. The right arrow $\rightarrow$ indicates that we use the left search engine to predict the right one, and $\leftarrow$ the converse.

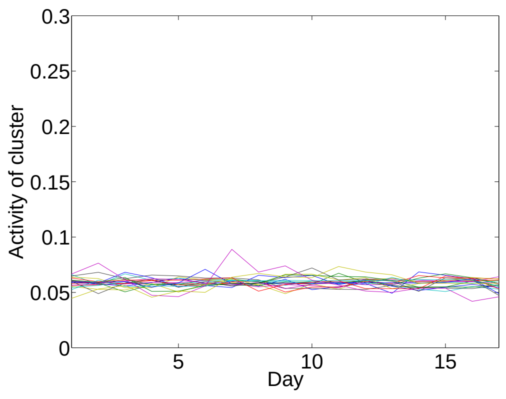
Firstly, we observe that the search results are mutually highly predictable for the TRENDS query set. This implies that the top results for these popular queries for Google and Bing are very similar. The same behavior continues to be observed for the MANUAL query set, albeit Google results are somewhat less predictable from Bing results.

12.5.3 Sensitivity Analysis

One might wonder how sensitive are our conclusions to the fact that we analyzed the top-10 search results. To this end, we apply TENSORCOMPARE and CROSSLEARNCOM-



(a) TRENDS query set



(b) MANUAL query set

Figure 12.5: **Temporal profile of latent clusters given by TENSORCOMPARE**: The y-axis corresponds to the membership of the particular day to the cluster of interest. For both query sets, the temporal profile of all clusters is approximately constant over time. In particular, each value for TRENDS is $\approx 1/21$ and for MANUAL it is $\approx 1/17$. As stated in Section 12.3.1, this indicates that both Bing and Google returned stable results. Due to this uniformity, we overplot all clusters, without making any distinctions.

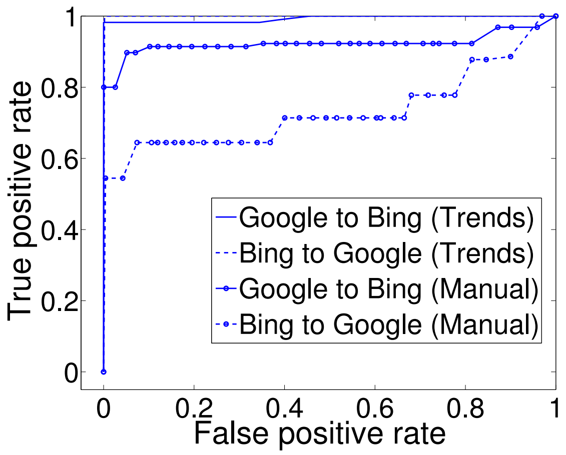


Figure 12.6: **ROC curves produced by CrossLearnComparefor top-10 results** (higher is better in terms of classification accuracy). If two search engines were completely mutually predictable, the ROC curve would be exactly on the $(0,0)-(0,1)$ and $(0,1)-(1,0)$ lines. Conversely, if two search engines were completely mutually unpredictable, the ROC curve would lie on the $(0,0)-(1,0)$ and $(1,0)-(1,1)$ lines. Finally, when the classifier is random, the curve would lie on the 45-degree line.

PARE to the top-5, top-3, as well top-1 search results, for both TRENDS and MANUAL query sets. Figure 12.7 shows the results of this analysis for TENSORCOMPARE. For CROSSLEARNCOMPARE, we omit ROC curves to conserve space and only show the AUC values in Table 12.3.

We see that our earlier findings are overall robust and consistent with the ones presented here. A few specific remarks follow:

- For larger values of k, the clusters for top-k results have greater similarity, for both Google and Bing. This behavior is as expected since a search result which is not included in top-k of one search engine might be in its $k+1$th position.
- For TRENDS queries, for values of k smaller than 10, it is easier to predict Google's results using a classifier trained on Bings' results than vice versa. It indicates that Bing exhibits relatively greater diversity in the results in earlier positions for these queries.
- For MANUAL queries, on the other hand, we see an inverse trend. It indicates that Google's unique results for these queries are concentrated in the early positions.
- For the single top result, even though there is similarity, the top result is not necessarily the same (but the manual inspection reveals that the top result of one is almost always present in the top-5 of the other)

12.6 Overlap Between Web and Social Results

We next present the results of comparing search results of Social Pulse first to that of Google and then Bing.

12.6.1 Social Pulse versus Google

Figure 12.8 and Table 12.4 show the results. We see from Figs. 12.8(a), 12.8(b):

1. There exists a number of results exclusive to either search engine as indicated by multiple points around $(0, 1)$ and $(1, 0)$.
2. For the non-exclusive results, the points are not concentrated on $(0.5, 0.5)$ (which would have indicated similar results), but are rather spread out.

This suggests that Social Pulse and Google provide distinctive results to a great extent.

For the TRENDS dataset in Fig. 12.8(a), there is a cloud of clusters around $(0.7, 0.3)$, which indicates that Google has greater participation in these results than Social Pulse. Figure 12.8(c) and AUC in Table 12.4 also show that using Google to predict Social Pulse works relatively better than the converse for this dataset. This asymmetry suggests that the Twitter users might not retweet much the readily-available, main-stream content on popular topics.

In contrast, for the MANUAL dataset in Fig. 12.8(b), the non-exclusive points are relatively more dispersed along the line that connects $(0, 1)$ and $(1, 0)$ and there are clusters in which Social Pulse is more prominent. We also find that now predicting Google using Social Pulse works better than the converse (Figs. 12.8(c) and 12.8(d)). Collectively, they quantitatively validate the intuition that social networks might have content very different from that indexed by web search engines for non-head queries.

	TRENDS →	TRENDS ←	MANUAL →	MANUAL ←
Google- Social Pulse	0.86	0.64	0.42	0.78

Table 12.4: AUC for CROSSLEARNCOMPARE comparing Google and Social Pulse for top-10 results.

	TRENDS →	TRENDS ←	MANUAL →	MANUAL ←
Bing- Social Pulse	0.86	0.60	0.44	0.83

Table 12.5: AUC for CROSSLEARNCOMPARE comparing Bing and Social Pulse for top-10 results.

12.6.2 Social Pulse versus Bing

We repeated the preceding analysis, but by using Bing search results rather than Google this time. Figure 12.9 and Table 12.5 show the results. These results are qualitatively similar to those obtained using Google search results, which is not surprising given the earlier finding that Google and Bing have significant overlap in their search results. However, this sensitivity analysis employing another commercial search engine further reinforces the conclusion that social search can yield results quite different from the ones produced by the conventional Web search.

12.6.3 Query Level Analysis

In order to gain further insight into mutual predictability of web and social search, we looked at three queries that have the highest and lowest predictability for each search engine and query set, when using CROSSLEARNCOMPARE analysis. Tables 12.6 and 12.7 show the results with respect to Google; the insights gained were similar for Bing.

We see that the timely queries, like *World cup* or *gay marriage*, have high mutual predictability. Indeed, timeliness creates relevance; the same information gets retweeted and clicked a lot. Queries like *Maya Angelou* and *Albert Einstein* are also highly mutually predictable, in part because people tend to tweet quotes by them, which tend to surface to Web search results as well.

On the other hand, queries such as *globalization* and *poverty* have low predictability. These queries are informational queries with large scope. However, it seems that the content people retweet a lot for these queries is not the same as what is considered authoritative by the web search ranking algorithms. We shall see that the majority of users in our user study found the results by Social Pulse for these queries to be very informative. This suggests a potentially interesting use case of Social Pulse, where the user does not have a crystalized a-priori expectation of the results and the search engine returns a set of results that have been filtered socially.

12.6.4 Sensitivity Analysis

We repeated our analysis for top-5, top-3 and top-1 search results. The results for Bing exhibited the same trend as Google, so we focus on presenting the results for Google. Figures 12.10 and Table 12.8 show the results. Overall we observe that our results are consistent, in terms of showing small overlap between Google and Social Pulse.

We also carried out another experiment in which we took the bottom five results from the top-6 results produced by Social Pulse and treated them as if they were the top-5 results of Social Pulse. We then compared these results to Google's top-5 results. Through this experiment, we wanted to get a handle on the robustness of our conclusions to the variations in Social Pulse's ranking function and the errors in tweet selection. We again found that the trends were preserved. We omit showing actual data.

12.6.5 Consistency with the TRM method

Recall our overview of the TRM method [TRM11], given in Section 12.2. In order to study the consistency between our results with what one would obtain using the TRM method, we conducted another sensitivity experiment. We first apply tensor analysis to the Google and Social Pulse results to obtain their condensed representations. We then compute the centroids for the Google and the Social Pulse results topics, and for every result from Google and Social Pulse (for all queries and days), we compute its cosine similarity to each centroid. While calculating the centroids, we ignore topics that are shared between Google and Social Pulse and keep those that lie on the $(0, 1)$ and $(1, 0)$ points of the TENSORCOMPARE plots. We present the results of this experiments in Table 12.9.

We again see that Google results in both query sets are more similar to the Google centroid, and Social Pulse results to the Social Pulse centroid. This analysis, this time employing a different method, further reinforces the conclusion that the social search results can be quite different from the conventional Web search results.

	Google → Social Pulse	Social Pulse → Google
TRENDS	SpongeBob SquarePants Albert Einstein Tottenham Hotspur F.C.	Oprah Winfrey Maya Angelou Albert Einstein
MANUAL	self-driving car gay marriage San Francisco	World cup gay marriage World bank

Table 12.6: Queries exhibiting highest predictability.

	Google → Social Pulse	Social Pulse → Google
TRENDS	Honda Antibiotics Frozen	Game of Thrones Skrillex Martini
MANUAL	coup education globalization	coup iPhone poverty

Table 12.7: Queries exhibiting lowest predictability.

Google- Social Pulse	TRENDS →	TRENDS ←	MANUAL →	MANUAL ←
top-10	0.86	0.64	0.42	0.78
top-5	0.87	0.70	0.39	0.66
top-3	0.86	0.50	0.35	0.69
top-1	0.79	0.98	0.50	0.53

Table 12.8: AUC for CROSSLEARNCOMPARE comparing Google and Social Pulse for different number of top results.

		To Google centroid	To Social Pulse centroid
TRENDS	From Google result	0.20	0.10
	From Social Pulse result	0.05	0.10
		To Google centroid	To Social Pulse centroid
MANUAL	From Google result	0.22	0.10
	From Social Pulse result	0.05	0.11

Table 12.9: Similarity from centroids

12.7 User Study

So far, we have discovered that the results of Social Pulse are different from Google and Bing. However, one might wonder whether these different results are actually useful, particularly given the apprehension that the content found on Twitter is of low quality [ACG+10]. To this end, we conducted a user study on the Amazon Mechanical Turk platform, following the best practices recommended in [Tur11]

12.7.1 HIT Design

Taking cue from the relevance judgment literature [CCSV11], the HIT (Human Intelligence Task) presented to the users consists of a query and a text representing a search result. The users are asked to select whether 1) the text is not informative, 2) the text is informative, or 3) it is hard to tell. They are then asked to explain their answer; any HIT that did not provide this explanation is rejected. Figure 12.11 shows a sample HIT.

We used the phrase "informative" rather than "relevant" in the instructions, after some initial testing. The choice "not informative" was placed above the positive one to avoid biasing the user's response towards the positive answer. Requiring users to explain their answer turned out to be important: users were forced to have a well justified reason why they selected a particular answer, minimizing random responses and other forms of noise.

Considering budget for the study, a subset of the queries were used. Both TRENDS and MANUAL queries were included; the reader can see the complete list in Fig. 12.15. A HIT was created for every query and each of the top-10 search results for the query. We asked every HIT to be judged by ten users.

12.7.2 Inter-User Agreement

To ensure there is consistency in the judgments provided by the users, we measured the inter-user agreement using the *Fleiss' kappa* test [Fle71]. In a nutshell, Fleiss' kappa (κ) is a number that indicates the degree of agreement between judges that is statistically significant and not attainable by chance. Its maximum value (for perfect agreement) is 1, and where there is no agreement, it can also take negative values. In exploratory tasks such as ours, a value in the range of 0.2-0.4 shows reasonable agreement and confidence on the results.

Although we had sought 10 judgments for every HIT, the actual deployment yielded the number of good judgments ranging from 4-10. Table 12.10 shows the κ values for our user study. A column of this table shows the number of search results that were judged exactly by the corresponding number of users as well the κ value. Thus, the column 1 of this table indicates that for twelve of the search results each was judged by exactly four users. We observe that κ is reasonably good in all cases, signifying good inter-user agreement.

# Judges	4	5	6	7	8	9	10
# Results	12	70	92	91	51	25	5
κ	0.19	0.29	0.27	0.26	0.32	0.43	0.22

Table 12.10: Inter-user agreement. A column of this table provides the number of search results that were judged exactly by the corresponding number of users as well as the κ value.

12.7.3 Sanity Checks

To further increase our confidence in the conclusions we arrive at, we did two sanity checks: i) visual inspection of the tweets in the result sets, and ii) their quantitative evaluation. We give below the results of both.

12.7.3.1 Visual Inspection

We examined top tweets for which there was high agreement amongst the judges as well as those tweets that had split judgments. Figures 12.12–12.14 show the top tweets from the two categories for which we had eight judgments each. It is readily apparent from these figures that the users were quite diligent in arriving at their decisions.

Metric	Google	Social Pulse
Kincaid	11.3	7.1
ARI	13.5	9.5
Coleman-Liau	12.9	11.4
Flesch Index	52.9/100	71.2/100
Fog Index	14.2	10.0
SMOG-Grading	12.5	9.6
Lix	49.3 (school year 9)	42.2 (school year 7)

Table 12.11: Readability of results.

12.7.3.2 Readability of the Result Tweets

It is a common belief that tweets are usually of bad quality, containing a lot of misspellings and illegible terms. But does this belief hold water when we focus on highly retweeted tweets? To quantitatively answer this question, we put the tweets in our result sets through the unix `style` tool. Given a piece of text, this tool computes seven metrics that have been extensively discussed in the literature and applied in practice [DuB04].

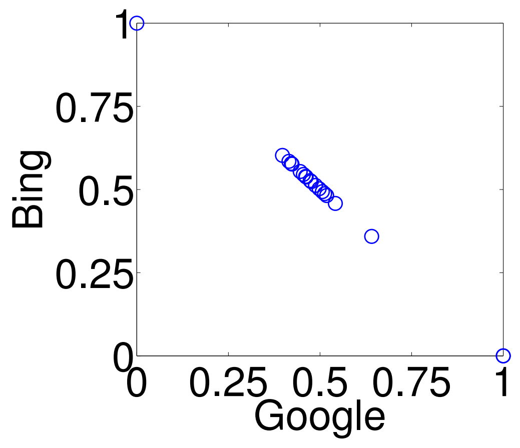
We conducted this study for Google and Social Pulse, for the same set of results that we use for the user study. Due to the nature of the `style` tool, we strip the snippets off any non alpha-numeric character, and we concatenate the snippets of each search engine into a longer passage, and apply `style` to it. The results are shown in Table 12.11.

It is not surprising that tweets score lower than Web snippets. The latter are derived from Web pages that are generally written much more formally whereas communication on Twitter is relatively informal. Note also that a lower value of a readability metric does not automatically imply lower understandability of the content. For example, the most popular novels are written at the 7th-grade level and people read for recreation texts that are two grades below their actual reading level [KB54]. Interestingly, we see from Table 12.11 that Lix pegs the readability of the result tweets at the 7th-grade level.

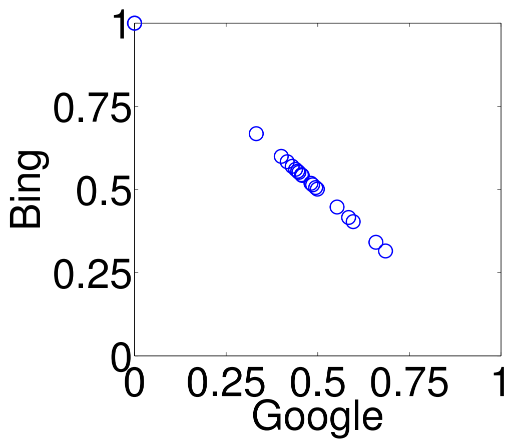
12.7.4 Results of the User Study

We can now finally present the results of our user study. Figure 12.15 summarizes them. We have plotted the usefulness index separately for each of the queries. For computing the usefulness index for a query, we consider every search result for a query for which we could get at least four judgments. We then check if a strict majority of users have judged the result to be informative for the given query. Note that "hard to tell" is treated as "not informative" for this purpose. The majority votes are then averaged over distinct search results for a specific query. Since the inter-user agreement is quite good according to Fleiss' kappa, the majority vote is a good indicator of the result quality.

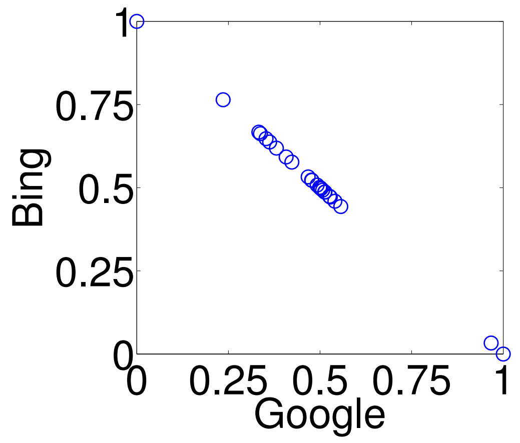
Overall, Fig. 12.15 demonstrates that most of the users found a large portion of Social Pulse's results informative with respect to the query in question, regardless of the query category (TRENDS or MANUAL). This finding is remarkable given the fact that the sole signal we use in order to discover and rank these results is the number of retweets.



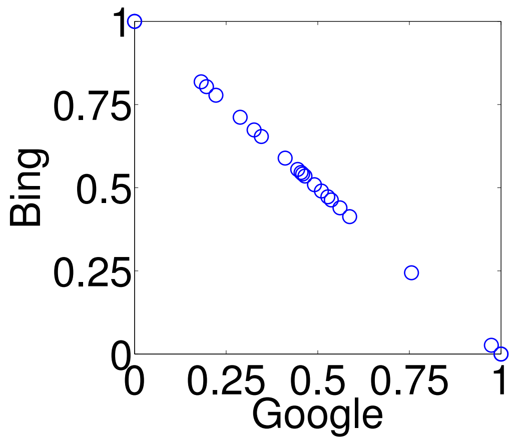
(a) TRENDS top-5



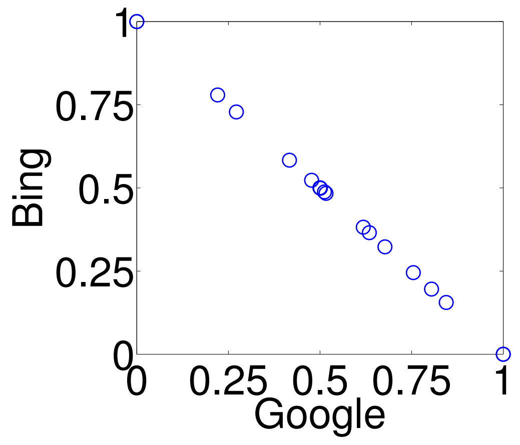
(b) MANUAL top-5



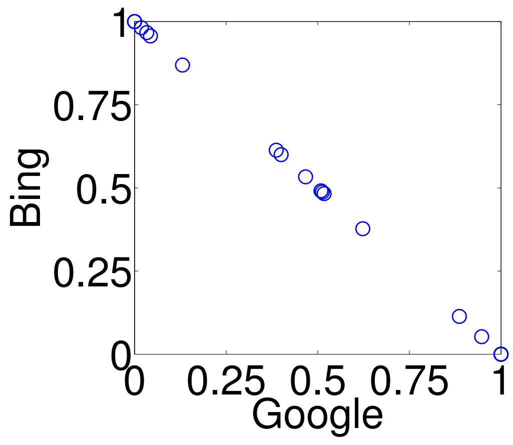
(c) TRENDS top-3



(d) MANUAL top-3

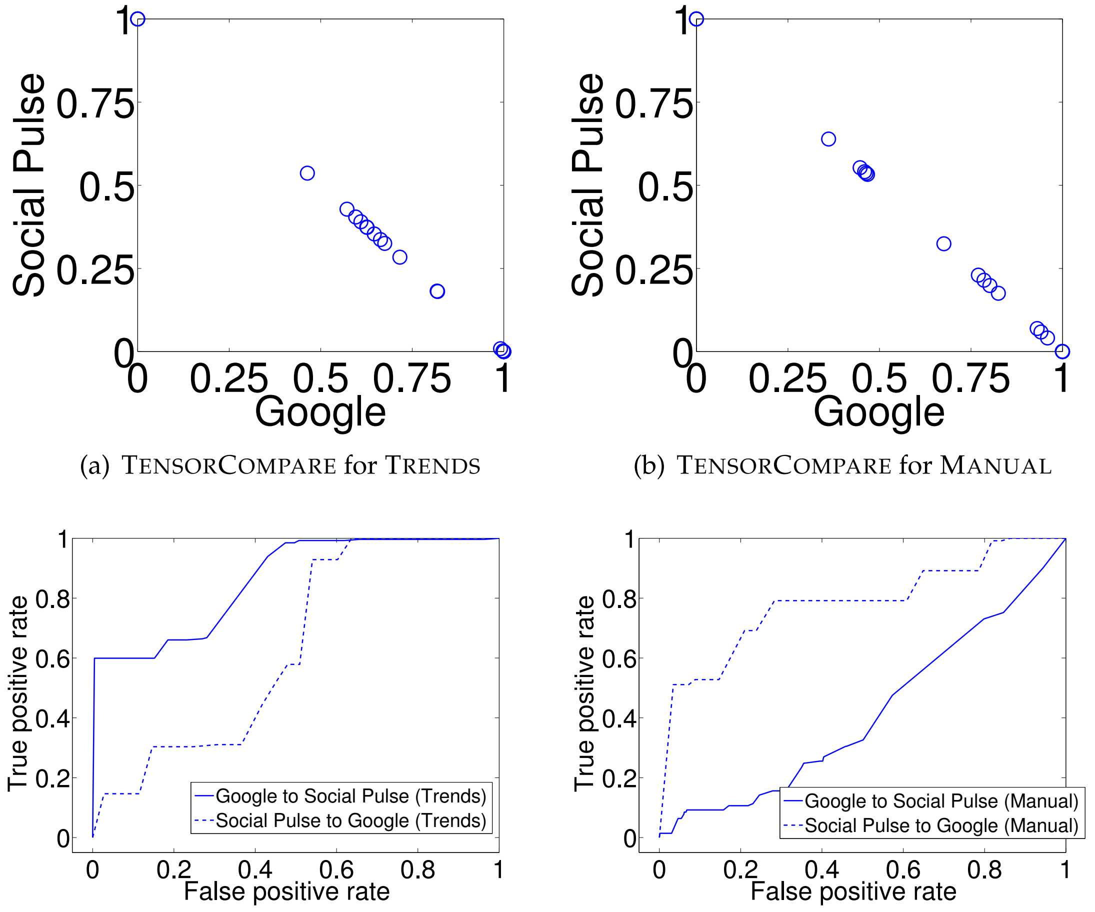


(e) TRENDS top-1



(f) MANUAL top-1

Figure 12.7: TENSORCOMPARE sensitivity with respect to the number of results included in the analysis



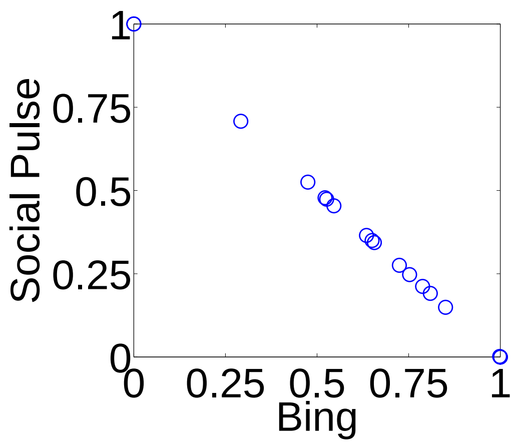
(a) TENSORCOMPARE for TRENDS

(b) TENSORCOMPARE for MANUAL

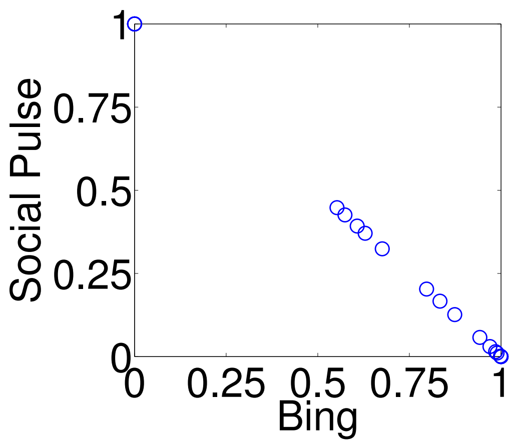
(c) CROSSLEARNCOMPARE for TRENDS

(d) CROSSLEARNCOMPARE for MANUAL

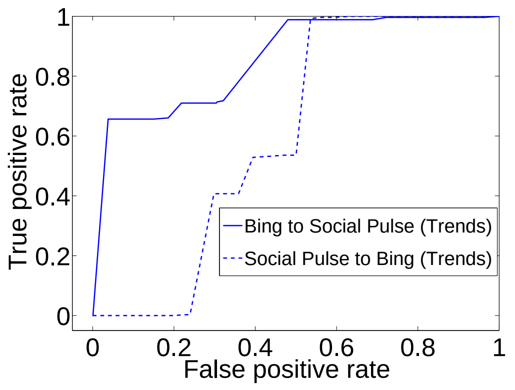
Figure 12.8: Social Pulse vs. Google for top-10 results



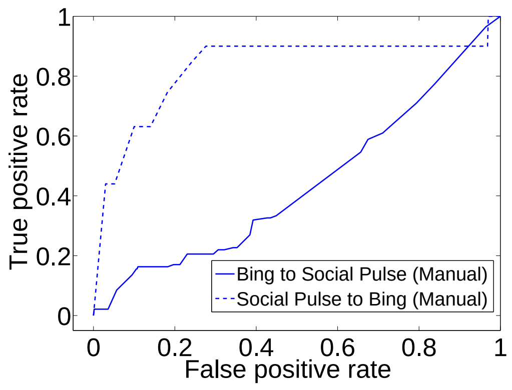
(a) TENSORCOMPARE for TRENDS



(b) TENSORCOMPARE for MANUAL

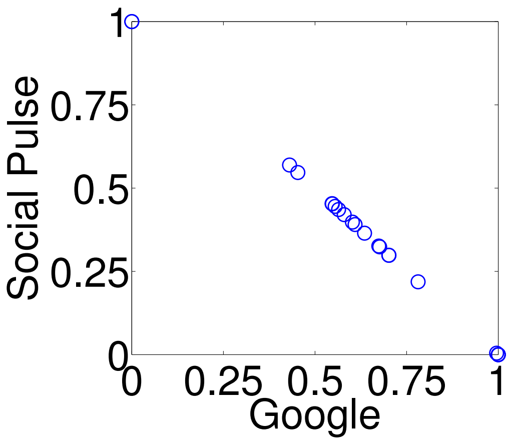


(c) CROSSLEARNCOMPARE for TRENDS

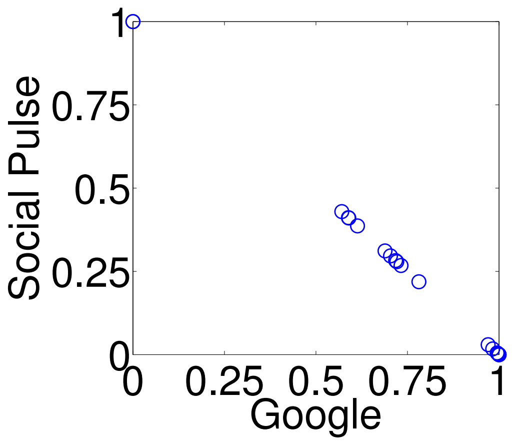


(d) CROSSLEARNCOMPARE for MANUAL

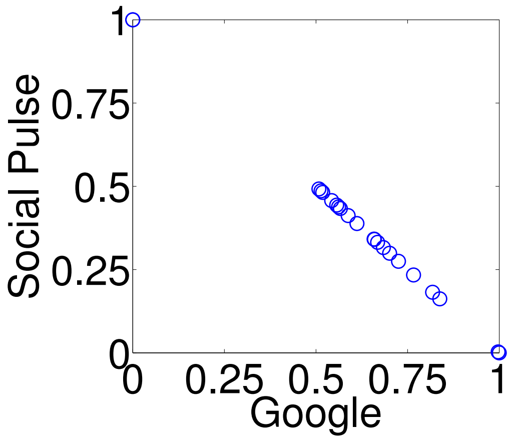
Figure 12.9: Social Pulse vs. Bing for top-10 results



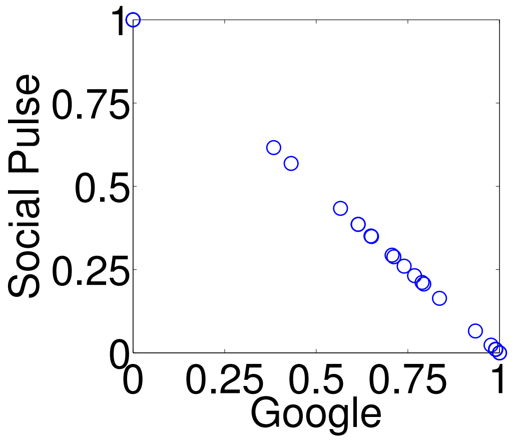
(a) TRENDS top-5



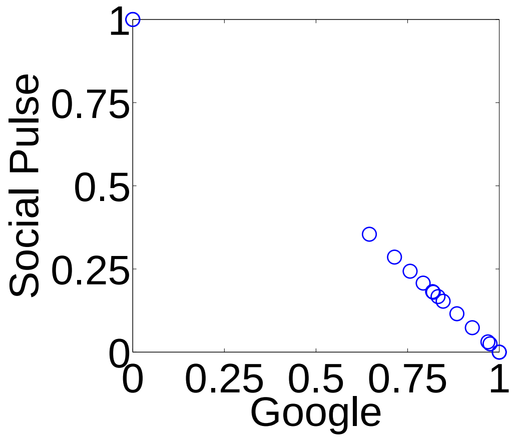
(b) MANUAL top-5



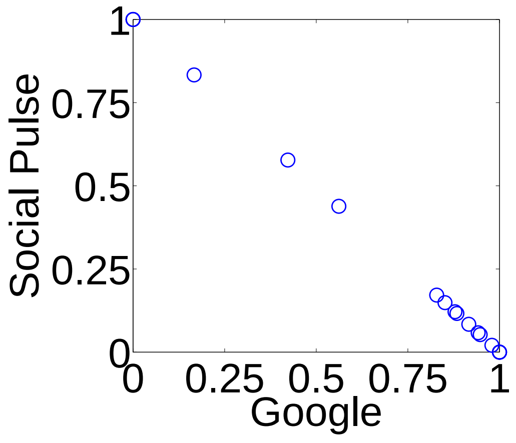
(c) TRENDS top-3



(d) MANUAL top-3

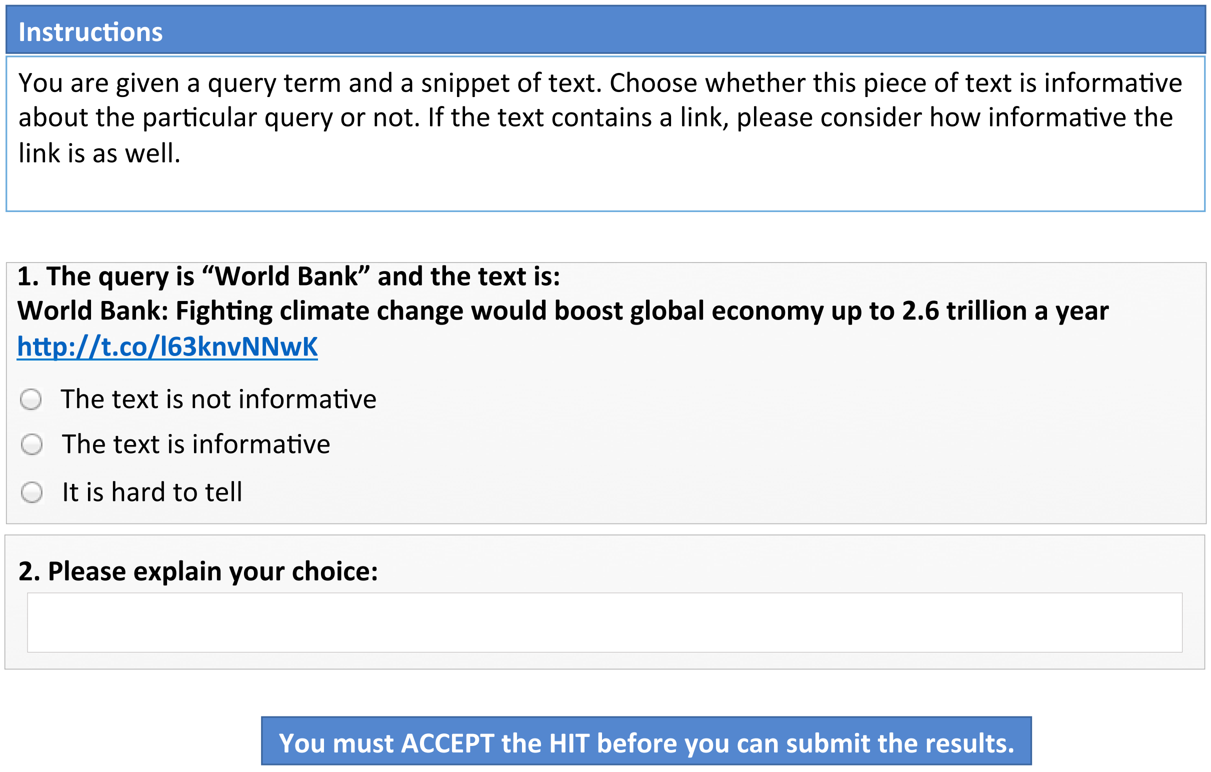


(e) TRENDS top-1



(f) MANUAL top-1

Figure 12.10: TENSORCOMPARE sensitivity with respect to the number of results included in the analysis



Instructions

You are given a query term and a snippet of text. Choose whether this piece of text is informative about the particular query or not. If the text contains a link, please consider how informative the link is as well.

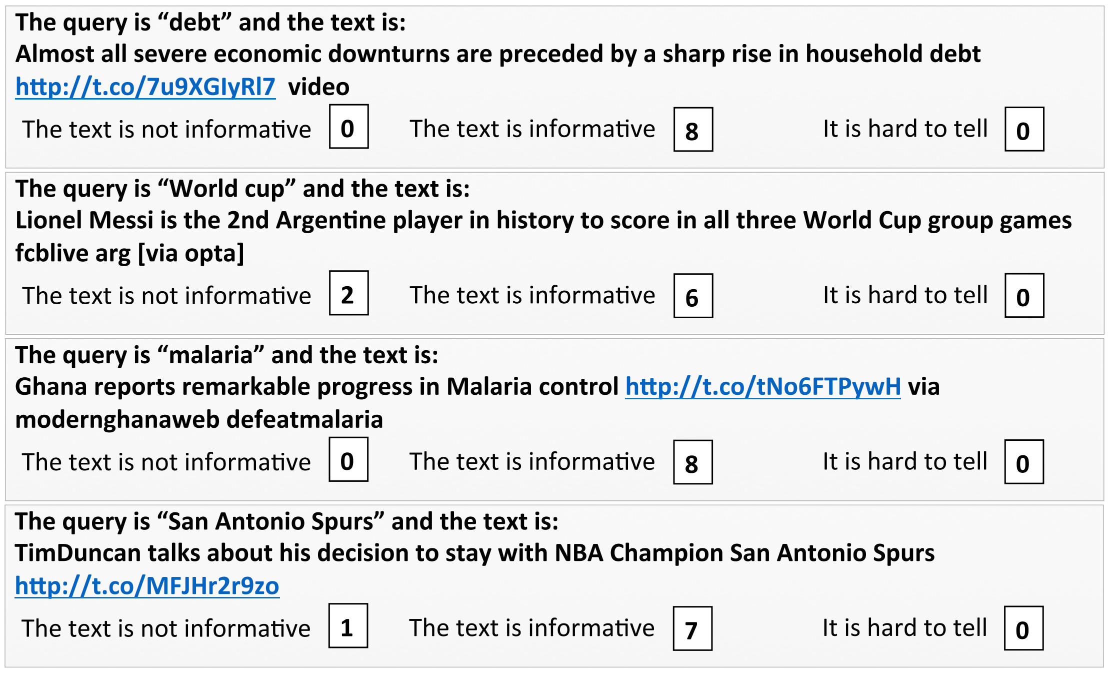
1. The query is "World Bank" and the text is:
World Bank: Fighting climate change would boost global economy up to 2.6 trillion a year http://t.co/l63knvNNwK

○ The text is not informative
○ The text is informative
○ It is hard to tell

2. Please explain your choice:

You must ACCEPT the HIT before you can submit the results.

Figure 12.11: A sample HIT



The query is "debt" and the text is:
Almost all severe economic downturns are preceded by a sharp rise in household debt http://t.co/7u9XGIyRl7 video

The text is not informative	0	The text is informative	8	It is hard to tell	0

The query is "World cup" and the text is:
Lionel Messi is the 2nd Argentine player in history to score in all three World Cup group games fcblive arg [via opta]

The text is not informative	2	The text is informative	6	It is hard to tell	0

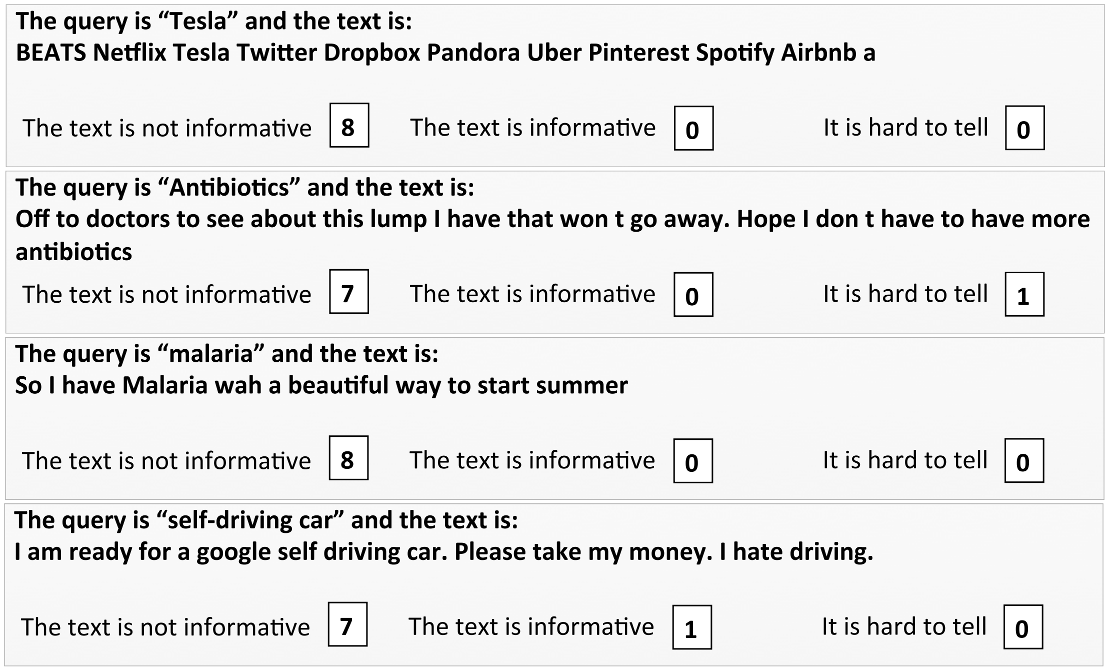
The query is "malaria" and the text is:
Ghana reports remarkable progress in Malaria control http://t.co/tNo6FTPywH via modernghanaweb defeatmalaria

The text is not informative	0	The text is informative	8	It is hard to tell	0

The query is "San Antonio Spurs" and the text is:
TimDuncan talks about his decision to stay with NBA Champion San Antonio Spurs http://t.co/MFJHr2r9zo

The text is not informative	1	The text is informative	7	It is hard to tell	0

Figure 12.12: Informative results with high judge agreement



The query is “Tesla” and the text is:
BEATS Netflix Tesla Twitter Dropbox Pandora Uber Pinterest Spotify Airbnb a

The text is not informative 8 The text is informative 0 It is hard to tell 0

The query is “Antibiotics” and the text is:
Off to doctors to see about this lump I have that won t go away. Hope I don t have to have more antibiotics

The text is not informative 7 The text is informative 0 It is hard to tell 1

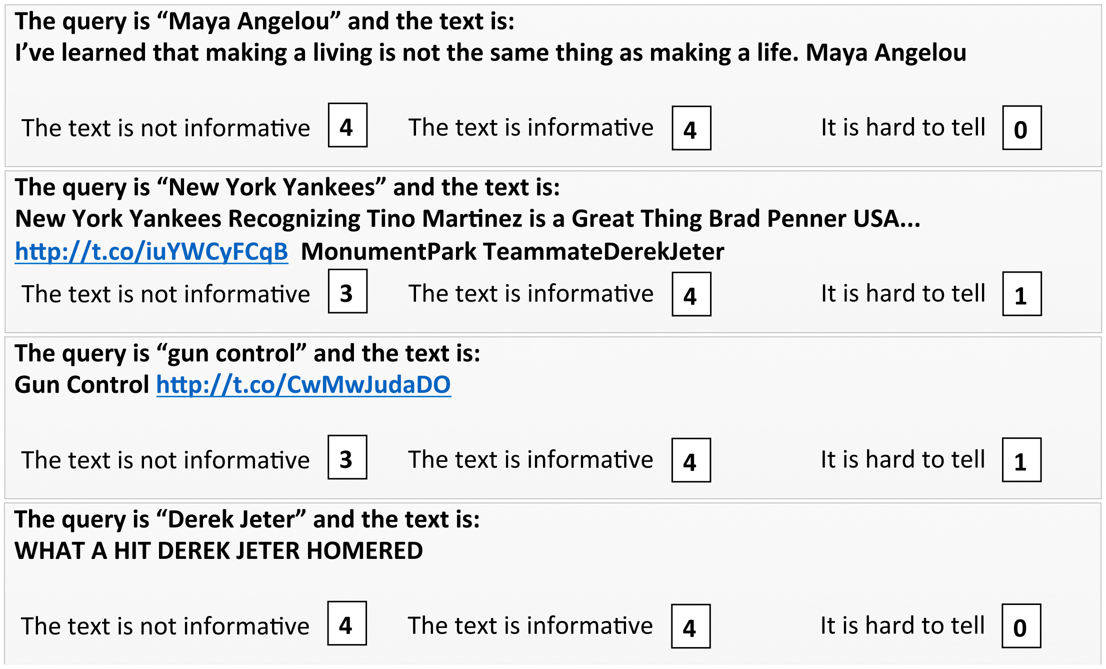
The query is “malaria” and the text is:
So I have Malaria wah a beautiful way to start summer

The text is not informative 8 The text is informative 0 It is hard to tell 0

The query is “self-driving car” and the text is:
I am ready for a google self driving car. Please take my money. I hate driving.

The text is not informative 7 The text is informative 1 It is hard to tell 0

Figure 12.13: Not informative results with high judge agreement



The query is “Maya Angelou” and the text is:
I’ve learned that making a living is not the same thing as making a life. Maya Angelou

The text is not informative 4 The text is informative 4 It is hard to tell 0

The query is “New York Yankees” and the text is:
New York Yankees Recognizing Tino Martinez is a Great Thing Brad Penner USA... http://t.co/iuYWCyFCqB MonumentPark TeammateDerekJeter

The text is not informative 3 The text is informative 4 It is hard to tell 1

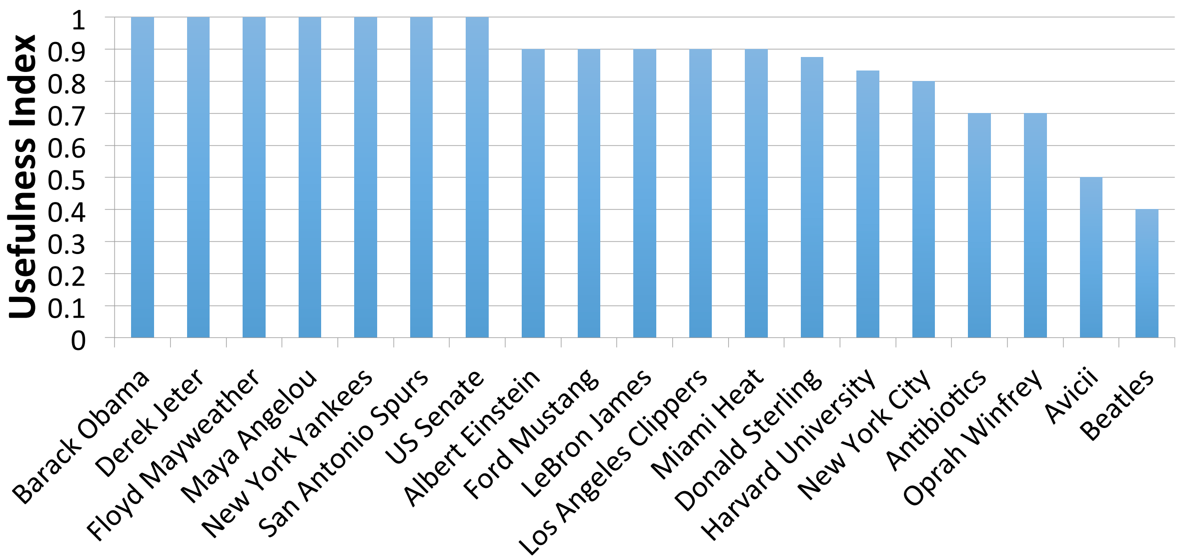
The query is “gun control” and the text is:
Gun Control http://t.co/CwMwJudaDO

The text is not informative 3 The text is informative 4 It is hard to tell 1

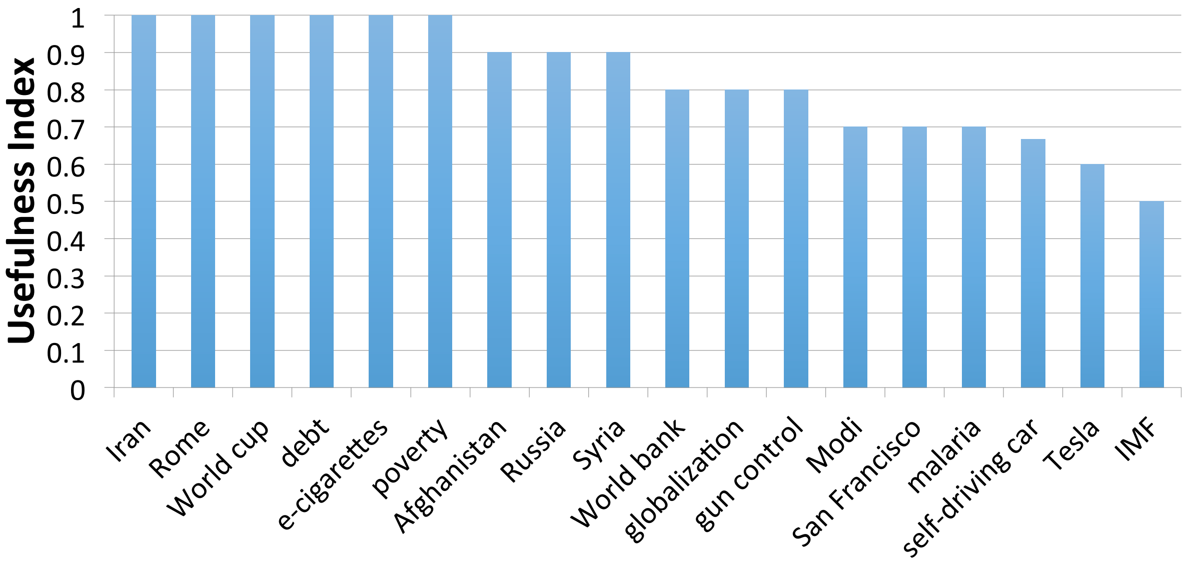
The query is “Derek Jeter” and the text is:
WHAT A HIT DEREK JETER HOMERED

The text is not informative 4 The text is informative 4 It is hard to tell 0

Figure 12.14: Results with poor judge agreement



(a) TRENDS



(b) MANUAL

Figure 12.15: Usefulness index of search results produced by Social Pulse for various queries

Part V

Conclusions and Future Directions

Chapter 13

Conclusions

13.1 Summary of Contributions

Multi-aspect data are ubiquitous, spanning a wide variety of real-world applications. In this thesis we introduce a set of tools that can *efficiently* and *effectively* model these multi-aspect data and improve knowledge discovery in the respective application.

13.1.1 Algorithms

The primary tool that we use to model multi-aspect data in this thesis is tensor decomposition. Our work has focused on

- *Scalable tensor decomposition*: Our work has focused on scaling up and parallelizing tensor decompositions. The algorithms we propose in this thesis have enabled processing of tensor datasets that the state of the art was unable to handle.
- *Scalable unsupervised quality assessment*: In addition to proposing fast and parallel tensor decomposition algorithms, our work has contributed on measuring the quality of those decompositions for much larger datasets (at least two orders of magnitude) than what the state of the art could, while additionally making more realistic assumptions for the distribution of the data.

13.1.2 Applications

- *Neurosemantics*: We pose the identification of semantically coherent brain regions as a novel Coupled Matrix-Tensor Factorization application and we demonstrate that such modeling can lead to discoveries that are known to Neuroscience in an *unsupervised* way; this offers confidence to the ability of this method to help and inform Neuroscientific research in scenarios where exploration is still under-way. Furthermore, we are the first to pose the estimation of the functional connectivity of the brain as a system identification problem, modeling the brain as a control system. Our model is able to identify concepts consistent with Neuroscience and can single out anomalies.

- *Social Networks and the Web*: We demonstrate that community detection and graph exploration greatly benefit by exploiting the inherent multi-aspect nature of the data. We achieve more accurate community detection by using different views of a social graph, and we gain better and more detailed insights on a social network by incorporating temporal and location information, as well as textual and lingual signals.
  Using our scalable tensor decomposition methods, we compute embeddings of NELL, a very large Knowledge Base, which enables efficient detection of synonymous or similar entities in the Knowledge Base. Furthermore, we introduce tools based on tensor analysis that measure the difference in the semantics of search results between search engines; we measure the similarity of Google and Bing and identify large overlap for popular queries. We, finally, propose a remedy to this overlap by demonstrating that social media based web search (using Twitter) offers diverse and useful search results.

13.2 Broader impact

The work presented in this thesis has already had significant impact which we categorize in 1) Academic Impact, 2) Impact to the research community through code release, and 3) Application Impact.

13.2.1 Academic Impact

The work presented here has been previously published in top-tier outlets in Data Mining and Signal Processing. We measure academic impact in terms of the citations, publication venue, and distinctions that our work has received.

- ParCube [PFS12] is the most cited paper of ECML-PKDD 2012 with over 60 citations at the time of writing, whereas the median number of citations for ECML-PKDD 2012 is 5.
- Our SDM 2016 paper introducing AutoTen [Pap16] won the NSF Travel Award and Best Student Paper Award.
- Turbo-SMT [PMS+14] was selected as one of the best papers of SDM 2014, and appeared in a special issue of the Statistical Analysis and Data Mining journal [PMS+16].
- Paracomp [SPF14] has appeared in the prestigious IEEE Signal Processing Magazine.
- GeBM [EFS+] is taught in class `CptS 595` at Washington State University.
- Our work in [AGP15a] was selected to appear in a special issue of the Journal of Web Science [AGP+16], as one of the best papers in the Web Science Track of WWW 2015.

13.2.2 Impact through code release

Towards enabling reproducibility and extendibility of our work, we have publicly released the source code for the algorithms presented in this thesis. Their impact to the research community is evidenced by the number of *distinct* universities, research organizations, and companies that have downloaded our code . Indicatively, as of June 25 2016:

- ParCube has been downloaded 125 times,
- GraphFuse has been downloaded 105 times
- GeBM has been downloaded 80 times
- Turbo-SMT has been downloaded 64 times
- Our algorithm in [PF15] has been downloaded 33 times

13.2.3 Application Impact

Finally, we measure the impact of the proposed algorithms to real-world applications:

- ParCube has been featured in an article by the Army Research Laboratory *Six Potential Game-Changers in Cyber Security: Towards Priorities in Cyber Science and Engineering* [KSM15] which was also presented as a keynote talk at the NATO Symposium on Cyber Security Science and Engineering 2014.
- Our work in [MWP+14] is deployed by the Institute for Information Industry in Taiwan, detecting real network intrusion attempts.

Chapter 14

Future Directions

As more field sciences are incorporating information technologies (with Computational Neuroscience being a prime example from a long list of disciplines), the need for scalable, efficient, and interpretable multi-aspect data mining algorithms will only increase. Below, we outline future research directions. First, we discuss our long-term and high-level vision on *Big Signal Processing for Data Science*, and we conclude with concrete future directions in Tensor Mining and Applications.

14.1 Long-Term Vision: Big Signal Processing for Data Science

The process of extracting useful and novel knowledge from big data in order to drive scientific discovery is the holy grail of data science. Consider the case where we view the knowledge extraction process through a signal processing lens: suppose that a transmitter (a physical or social phenomenon) is generating signals which describe aspects of the underlying phenomenon. An example signal is a time-evolving graph (a time-series of graphs) which can be represented as a tensor. The receiver (the data scientist in our case), combines all those signals with ultimate goal the reconstruction (and general understanding) of their generative process. The "communication channel" wherein the data are transmitted can play the role of the measurement process where loss of data occurs, and thus we have to account the channel estimation into our analysis, in order to reverse its effect. We may consider that the best way to transmit all the data to the receiver is to find the best compression or dimensionality reduction of those signals (e.g., *Compressed Sensing*). There may be more than one transmitters (if we consider a setting where the data are distributed across data centers) and multiple receivers, in which case privacy considerations come into play. In my work so far we have established two connections between Signal Processing and Data Science (Tensor Decompositions & System Identification in Chapter 10), contributing new results in both communities and have already had significant impact, demonstrated, for instance, by the amount of researchers using and extending my work. These two aforementioned connections are but instances of a vast landscape of opportunities for cross-pollination which will advance the state of the art of both fields and drive scientific discovery. We envision our future work as a three-way bridge between Signal Processing, Data Science, and high impact real-world applications.

14.2 Future Directions in Tensor Mining

The success that tensors have experienced in data mining during the last few years by no means indicates that all challenges and open problems have been addressed. Quite to the contrary, there exist challenges, some of which we summarize in the next few lines, which delineate very exciting future research directions:

- **Modeling space and time**: What is the best way to exploit spatial or temporal structure that exists in the data? We saw examples in [DGCW13] where the authors impose linear constraints that guide the decomposition according to prior knowledge of the spatial structure, [MHA$^+$08, MHM11] where the authors deal with temporal issues in brain data, and [STH$^+$08] where the authors use a wavelet decomposition to represent time. Is there a generic way to incorporate such modifications in a tensor model and enable it to handle spatio-temporal data effectively? Furthermore, another open problem in spatio-temporal data modeling is selecting the right granularity for the space and time modes.
- **Unsupervised model selection**: In a wide variety of applications, ground truth is not easy to obtain, however we need to have unsupervised means of understanding which tensor decomposition is more appropriate (e.g., PARAFAC vs TUCKER vs DEDICOM etc), and given a decomposition, what model order is most appropriate for the data.
- **Dealing with high-order data**: Many real-world applications involve data that can be represented as very high-order tensors. Works that use H-Tucker [PCVS15] have shown the utility of such approaches, and in the future, work on understanding and improving decompositions such as H-Tucker and Tensor-Train, in the context of data mining, will be very important.
- **Connections with Heterogeneous Information Networks**: In Data Mining, there exists a very rich line of work on Heterogeneous Information Networks (HINs), which are graphs between different types of nodes, connected with various types of edges. A HIN can be represented as a tensor, and in fact, a multi-view social network is such a HIN. In the HIN mining literature, there exist concepts such as the "Meta-path" [SHY$^+$11] which is a path within the network that traverses multiple types of nodes, in the same spirit as a random walk, aiming to find similar nodes in the network (and has also been used for clustering). Outlining connections between such works and tensor decompositions is a very interesting future direction that aims towards unifying different data mining approaches.

14.3 Future Directions in Applications

14.3.1 Application: Urban & Social Computing

Social and physical interactions of people in an urban environment, is an inherent multi-aspect process, that ties the physical and the on-line domains of human interaction. We plan to investigate human mobility patterns, e.g. through check-ins in on-line social networks, in combination with their social interactions and the content they create on-line, with specific emphasis on multi-lingual content which is becoming very prevalent due to population mobility. We also intend to develop anomaly detection which can point to fraud (e.g., people buying fake followers for their account). Improving user experience through identifying normal and anomalous patterns in human geo-social activity is an extremely important problem, both in terms of funding and research interest, as well as implications on revolutionizing modern societies.

14.3.2 Application: Neurosemantics

In the search for understanding how semantic information is processed by the brain, we are planning to broaden the scope of the Neurosemantics applications, considering aspects such as *language*: are same concepts in different languages mapped in the same way in the brain? Are cultural idiosyncrasies reflected on the way that speakers of different languages represent information? Furthermore, we may consider more complex forms of stimuli (such as phrases, images, and video) and richer sources of semantic information, e.g, from Knowledge on the Web. There is profound scientific interest in answering the above research questions a fact also reflected on how well funded of a research area this is (e.g., see Brain Initiative http://braininitiative.nih.gov/)

14.3.3 Application: Knowledge on the Web

Knowledge bases are ubiquitous, providing taxonomies of human knowledge and facilitating web search. Many knowledge bases are extracted automatically, and as such they are noisy and incomplete. Furthermore, web content exists in multiple languages, which is inherently imbalanced and may result in imbalanced knowledge bases, consequently leading to skewed views of web knowledge per language. We plan to pursue work on web knowledge, devising and developing techniques that combine multiple, multilingual, structured (e.g., knowledge bases) and unstructured (e.g., plain text) sources of information on the web, aiming for high quality knowledge representation, as well as enrichment and curation of knowledge bases.

Bibliography

[AABB$^+$07] Evrim Acar, Canan Aykut-Bingol, Haluk Bingol, Rasmus Bro, and Bülent Yener. Multiway analysis of epilepsy tensors. *Bioinformatics*, 23(13):i10–i18, 2007. 16, 43, 162

[AB00] C.A. Andersson and R. Bro. The n-way toolbox for matlab. *Chemometrics and Intelligent Laboratory Systems*, 52(1):1–4, 2000. 61, 135

[ABK16] Woody Austin, Grey Ballard, and Tamara G. Kolda. Parallel tensor compression for large-scale scientific data. In *IPDPS'16: Proceedings of the 30th IEEE International Parallel & Distributed Processing Symposium*, May 2016. 57, 58

[ACG$^+$09] Frederico AC Azevedo, Ludmila RB Carvalho, Lea T Grinberg, José Marcelo Farfel, Renata EL Ferretti, Renata EP Leite, Roberto Lent, Suzana Herculano-Houzel, et al. Equal numbers of neuronal and non-neuronal cells make the human brain an isometrically scaled-up primate brain. *Journal of Comparative Neurology*, 513(5):532–541, 2009. 164

[ACG$^+$10] Omar Alonso, Chad Carson, David Gerster, Xiang Ji, and Shubha U Nabar. Detecting uninteresting content in text streams. In *SIGIR Crowdsourcing for Search Evaluation Workshop*, 2010. 203, 209, 218

[A**c**CKY05] Evrim Acar, Seyit A **c**Camtepe, Mukkai S Krishnamoorthy, and Bülent Yener. Modeling and multiway analysis of chatroom tensors. In *Intelligence and Security Informatics*, pages 256–268. Springer, 2005. 39

[ADK11] Evrim Acar, Daniel M Dunlavy, and Tamara G Kolda. A scalable optimization approach for fitting canonical tensor decompositions. *Journal of Chemometrics*, 25(2):67–86, 2011. 24, 96

[ADKM10] Evrim Acar, Daniel M Dunlavy, Tamara G Kolda, and Morten Mørup. Scalable tensor factorizations with missing data. In *SDM*, pages 701–712. SIAM, 2010. 25

[AGH$^+$14] Animashree Anandkumar, Rong Ge, Daniel Hsu, Sham M Kakade, and Matus Telgarsky. Tensor decompositions for learning latent variable models. *The Journal of Machine Learning Research*, 15(1):2773–2832, 2014. 40, 43, 204

[AGP15a] Rakesh Agrawal, Behzad Golshan, and Evangelos Papalexakis. A study of distinctiveness in web results of two search engines. In *24th international conference on World Wide Web, Web Science Track*. ACM, May 2015. 7, 8, 42, 199, 232

[AGP15b] Rakesh Agrawal, Behzad Golshan, and Evangelos Papalexakis. Whither social networks for web search? In *21st ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, Sydney, Australia, August 2015. 7, 42, 199

[AGP$^+$16] Rakesh Agrawal, Behzad Golshan, Evangelos Papalexakis, et al. Overlap in the web search results of google and bing. *The Journal of Web Science*, 2(2):17–30, 2016. 8, 232

[AGR$^+$12] E. Acar, G. Gurdeniz, M.A. Rasmussen, D. Rago, L.O. Dragsted, and R. Bro. Coupled matrix factorization with sparse factors to identify potential biomarkers in metabolomics. In *IEEE ICDM Workshops*, pages 1–8. IEEE, 2012. 17

[AKD11] Evrim Acar, Tamara G Kolda, and Daniel M Dunlavy. All-at-once optimization for coupled matrix and tensor factorizations. *arXiv preprint arXiv:1105.3422*, 2011. 17, 35, 86, 87, 95, 96, 97, 99, 100

[ALR$^+$13] Evrim Acar, Anders J Lawaetz, Morten Rasmussen, Rasmus Bro, et al. Structure-revealing data fusion model with applications in metabolomics. In *Engineering in Medicine and Biology Society (EMBC), 2013 35th Annual International Conference of the IEEE*, pages 6023–6026. IEEE, 2013. 36

[AM01] Javed A Aslam and Mark Montague. Models for metasearch. In *24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 276–284. ACM, 2001. 203

[And83] J.R. Anderson. A spreading activation theory of memory. *Journal of verbal learning and verbal behavior*, 22(3):261–95, 1983. 182

[APG$^+$14] Miguel Araujo, Spiros Papadimitriou, Stephan Günnemann, Christos Faloutsos, Prithwish Basu, Ananthram Swami, Evangelos E Papalexakis, and Danai Koutra. Com2: Fast automatic discovery of temporal ('comet') communities. In *Advances in Knowledge Discovery and Data Mining*, pages 271–283. Springer, 2014. 6, 8, 39, 41, 51, 56, 153

[APK$^+$11] Demetris Antoniades, Iasonas Polakis, Georgios Kontaxis, Elias Athanasopoulos, Sotiris Ioannidis, Evangelos P Markatos, and Thomas Karagiannis. we.b: The web of short URLs. In *20th international conference on World Wide Web*, pages 715–724. ACM, 2011. 202, 210

[APR$^+$14] Evrim Acar, Evangelos E Papalexakis, Morten A Rasmussen, Anders J Lawaetz, Mathias Nilsson, Rasmus Bro, et al. Structure-revealing data fusion. *BMC bioinformatics*, 15(1):239, 2014. 36

[APY12] E. Acar, G.E. Plopper, and B. Yener. Coupled analysis of in vitro and histology tissue samples to quantify structure-function relationship. *PloS one*, 7(3):e32227, 2012. 17

[ARS$^+$13] Evrim Acar, Morten Arendt Rasmussen, Francesco Savorani, Tormod Næs, and Rasmus Bro. Understanding data fusion within the framework of coupled matrix and tensor factorizations. *Chemometrics and Intelligent Laboratory Systems*, 129:53–63, 2013. 86

[ATMF12] Leman Akoglu, Hanghang Tong, Brendan Meeder, and Christos Faloutsos. PICS: Parameter-free identification of cohesive subgroups in large attributed graphs. In *SDM*, pages 439–450, 2012. 190, 192

[AW10] Charu C. Aggarwal and Haixun Wang, editors. *Managing and Mining Graph Data*, volume 40 of *Advances in Database Systems*. Springer, 2010. 188

[AY09] Evrim Acar and Bülent Yener. Unsupervised multiway data analysis: A literature survey. *Knowledge and Data Engineering, IEEE Transactions on*, 21(1):6–20, 2009. 21

[BA98] Rasmus Bro and Claus A Andersson. Improving the speed of multiway algorithms: Part ii: Compression. *Chemometrics and Intelligent Laboratory Systems*, 42(1):105–113, 1998. 51, 54, 56, 100

[BB98] Krishna Bharat and Andrei Broder. A technique for measuring the relative size and overlap of public web search engines. *Computer Networks and ISDN Systems*, 30(1):379–388, 1998. 201

[BBM07] Arindam Banerjee, Sugato Basu, and Srujana Merugu. Multi-way clustering on relation graphs. In *SDM*, volume 7, pages 145–156. SIAM, 2007. 17, 85, 87

[BCG11] Michele Berlingerio, Michele Coscia, and Fosca Giannotti. Finding redundant and complementary communities in multidimensional networks. In *CIKM*, 2011. 189

[BD96] Paul E Buis and Wayne R Dyksen. Efficient vector and parallel manipulation of tensor products. *ACM Transactions on Mathematical Software (TOMS)*, 22(1):18–23, 1996. 134, 135, 141

[BD06] Christopher D Brown and Herbert T Davis. Receiver operating characteristics curves and related decision measures: A tutorial. *Chemometrics and Intelligent Laboratory Systems*, 80(1):24–38, 2006. 213

[BD11] Jeffrey R Binder and Rutvik H Desai. The neurobiology of semantic memory. *Trends in cognitive sciences*, 15(11):527–36, November 2011. 177

[BGHS12] Brigitte Boden, Stephan Günnemann, Holger Hoffmann, and Thomas Seidl. Mining coherent subgraphs in multi-layer graphs with edge labels. In *KDD*, 2012. 189

[BGK13] Jonas Ballani, Lars Grasedyck, and Melanie Kluge. Black box approximation of tensors in hierarchical tucker format. *Linear algebra and its applications*, 438(2):639–657, 2013. 31

[BGL$^+$12] Michael Busch, Krishna Gade, Brian Larson, Patrick Lok, Samuel Luckenbill, and Jimmy Lin. Earlybird: Real-time search at Twitter. In *IEEE 28th International Conference on Data Engineering*, pages 1360–1369. IEEE, 2012. 203

[BHK06] B.W. Bader, R.A. Harshman, and T.G. Kolda. Temporal analysis of social networks using three-way dedicom. *Sandia National Laboratories TR SAND2006-2161*, 2006. 40, 80, 130

[BHK07] Brett W Bader, Richard A Harshman, and Tamara G Kolda. Temporal analysis of semantic graphs using asalsan. In *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*, pages 33–42. IEEE, 2007. 29, 30, 31, 40

[BI04] Judit Bar-Ilan. Search engine ability to cope with the changing web. In *Web dynamics*, pages 195–215. Springer, 2004. 202

[BK03] Rasmus Bro and Henk AL Kiers. A new efficient method for determining the number of components in parafac models. *Journal of chemometrics*, 17(5):274–286, 2003. 38, 130, 132, 140, 152, 153

[BK07] Brett W. Bader and Tamara G. Kolda. Efficient MATLAB computations with sparse and factored tensors. *SIAM Journal on Scientific Computing*, 30(1):205–231, December 2007. 52, 56, 57, 107, 130, 140

[BK$^+$15] Brett W. Bader, Tamara G. Kolda, et al. Matlab tensor toolbox version 2.6. Available online, February 2015. 52, 61, 72, 97, 100, 107, 135, 141, 147, 212

[BKML13] Rabia Batool, Asad Masood Khattak, Jahanzeb Maqbool, and Sungyoung Lee. Precise tweet classification and sentiment analysis. In *IEEE/ACIS 12th International Conference on Computer and Information Science*, pages 461–466. IEEE, 2013. 209

[BKP$^+$14] Alex Beutel, Abhimanu Kumar, Evangelos Papalexakis, Partha Pratim Talukdar, Christos Faloutsos, and Eric P Xing. Flexifact: Scalable flexible factorization of coupled tensors on hadoop. In *SIAM SDM*, 2014. 4, 54, 55, 56

[BNJ03] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003. 43, 204

[BPC$^+$11] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011. 54

[Bre78] J.W. Brewer. Kronecker products and matrix calculus in system theory. *IEEE Trans. on Circuits and Systems*, 25(9):772–781, 1978. 13, 113

[Bro97] R. Bro. Parafac. tutorial and applications. *Chemometrics and intelligent laboratory systems*, 38(2):149–171, 1997. 129, 205

[Bro98] Rasmus Bro. *Multi-way analysis in the food industry: models, algorithms, and applications*. PhD thesis, 1998. 38, 94, 130, 132

[Bro02] Andrei Broder. A taxonomy of web search. *ACM Sigir forum*, 36(2):3–10, 2002. 209

[BS98] R. Bro and N.D. Sidiropoulos. Least squares regression under unimodality and non-negativity constraints. *Journal of Chemometrics*, 12:223–247, 1998. 110

[BSG99] R Bro, ND Sidiropoulos, and GB Giannakis. A fast least squares algorithm for separating trilinear mixtures. In *Int. Workshop Independent Component and Blind Signal Separation Anal*, pages 11–15, 1999. 79, 100, 112

[BYKS09] Ziv Bar-Yossef, Idit Keidar, and Uri Schonfeld. Do not crawl in the DUST: different urls with similar text. *ACM Transactions on the Web*, 3(1):3, 2009. 202, 210

[CBKA07] P.A. Chew, B.W. Bader, T.G. Kolda, and A. Abdelali. Cross-language information retrieval using parafac2. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 143–152. ACM, 2007. 37, 42

[CC70a] J Douglas Carroll and Jih-Jie Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of âĂIJeckart-youngâĂİ decomposition. *Psychometrika*, 35(3):283–319, 1970. 14

[CC70b] J.D. Carroll and J.J. Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of âĂIJEckart-YoungâĂİ decomposition. *Psychometrika*, 35(3):283–319, 1970. 110, 111

[CC10] Cesar F Caiafa and Andrzej Cichocki. Generalizing the column–row matrix decomposition to multi-way arrays. *Linear Algebra and its Applications*, 433(3):557–573, 2010. 57, 58

[CCSV11] Charles L. A. Clarke, Nick Craswell, Ian Soboroff, and Ellen M. Voorhees. Overview of the TREC 2011 web track. Technical report, NIST, 2011. 200, 218

[CFC15] Jeremy E Cohen, Rodrigo Cabral Farias, and Pierre Comon. Fast decomposition of large nonnegative tensors. *Signal Processing Letters, IEEE*, 22(7):862–866, 2015. 51, 54, 56

[Cha04] Deepayan Chakrabarti. Autopart: Parameter-free graph partitioning and outlier detection. In *PKDD*, pages 112–124, 2004. 190, 192

[CK01] Anne Condon and Richard M. Karp. Algorithms for graph partitioning on the planted partition model. *Random Struct. Algorithms*, 18(2):116–140, 2001. 194

[CK12] Eric C Chi and Tamara G Kolda. On tensors, sparsity, and nonnegative factorizations. *SIAM Journal on Matrix Analysis and Applications*, 33(4):1272–1299, 2012. 26, 140, 141, 142, 148, 149, 212

[CL75] Allan M. Collins and Elizabeth F. Loftus. A spreading-activation theory of semantic processing. *Psychological Review*, 82(6):407–428, 1975. 182

[CMDL$^+$15] Andrzej Cichocki, Danilo Mandic, Lieven De Lathauwer, Guoxu Zhou, Qibin Zhao, Cesar Caiafa, and Huy Anh Phan. Tensor decompositions for signal processing applications: From two-way to multiway component analysis. *Signal Processing Magazine, IEEE*, 32(2):145–163, 2015. 21

[CMP11] Carlos Castillo, Marcelo Mendoza, and Barbara Poblete. Information credibility on twitter. In *20th international conference on World wide web*, pages 675–684. ACM, 2011. 203

[CO12] L. Chiantini and G. Ottaviani. On generic identifiability of 3-tensors of small rank. *SIAM. J. Matrix Anal. & Appl.*, 33(3):1018–1037, 2012. 15, 22, 111, 112, 121, 146

[CPK80] J Douglas Carroll, Sandra Pruzansky, and Joseph B Kruskal. Candelinc: A general approach to multidimensional analysis of many-way arrays with linear constraints on parameters. *Psychometrika*, 45(1):3–24, 1980. 51, 100

[CR96] Heting Chu and Marilyn Rosenthal. Search engines for the world wide web: A comparative study and evaluation methodology. In *American Society for Information Science*, volume 33, pages 127–135, 1996. 201

[CV14] Joon Hee Choi and S. Vishwanathan. Dfacto: Distributed factorization of tensors. In Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 1296–1304. Curran Associates, Inc., 2014. 52, 56

[CYM13] Kai-Wei Chang, Wen-tau Yih, and Christopher Meek. Multi-relational latent semantic analysis. In *EMNLP*, pages 1602–1612, 2013. 43

[CYYM14] Kai-Wei Chang, Wen-tau Yih, Bishan Yang, and Christopher Meek. Typed tensor decomposition of knowledge bases for relation extraction. In

Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1568–1579, 2014. 43

[CZPA09] Andrzej Cichocki, Rafal Zdunek, Anh Huy Phan, and Shun-ichi Amari. *Nonnegative matrix and tensor factorizations: applications to exploratory multi-way data analysis and blind source separation*. John Wiley & Sons, 2009. 21, 25, 162

[DAK$^+$14] André LF De Almeida, Alain Y Kibangou, et al. Distributed large-scale tensor decomposition. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, 2014. 54, 56

[dCHR08] Joao Paulo CL da Costa, Martin Haardt, and F Romer. Robust methods based on the hosvd for estimating the model order in parafac models. In *Sensor Array and Multichannel Signal Processing Workshop, 2008. SAM 2008. 5th*, pages 510–514. IEEE, 2008. 38, 130, 132

[DDF$^+$90] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, September 1990. 42, 160

[DE03] D.L. Donoho and M. Elad. Optimally sparse representation in general (nonorthogonal) dictionaries via minimization. *Proc. Nat. Acad. Sci.*, 100(5):2197–2202, 2003. 122

[DGCW13] Ian Davidson, Sean Gilpin, Owen Carmichael, and Peter Walker. Network discovery via constrained tensor analysis of fmri data. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 194–202. ACM, 2013. 44, 162, 181, 236

[DJQ$^+$10] Yajuan Duan, Long Jiang, Tao Qin, Ming Zhou, and Heung-Yeung Shum. An empirical study on learning to rank of tweets. In *23rd International Conference on Computational Linguistics*, pages 295–303. Association for Computational Linguistics, 2010. 203

[DKA11] Daniel M Dunlavy, Tamara G Kolda, and Evrim Acar. Temporal link prediction using matrix and tensor factorizations. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 5(2):10, 2011. 140

[DKM$^+$06] P. Drineas, R. Kannan, M.W. Mahoney, et al. Fast monte carlo algorithms for matrices iii: Computing a compressed approximate matrix decomposition. *SIAM Journal on Computing*, 36(1):184, 2006. 57, 61, 62

[DLDMV00] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. A multilinear singular value decomposition. *SIAM journal on Matrix Analysis and Applications*, 21(4):1253–1278, 2000. 15, 27, 100

[DM96] Wei Ding and Gary Marchionini. A comparative study of web search service performance. In *ASIS Annual Meeting*, volume 33, pages 136–42. ERIC, 1996. 201

[DuB04] W.H. DuBay. *The principles of readability*. Impact Information, 2004. 220

[DZK$^+$10] Anlei Dong, Ruiqiang Zhang, Pranam Kolari, Jing Bai, Fernando Diaz, Yi Chang, Zhaohui Zheng, and Hongyuan Zha. Time is of the essence: improving recency ranking using twitter data. In *Proceedings of the 19th international conference on World wide web*, pages 331–340. ACM, 2010. 205

[EAC15] Beyza Ermics, Evrim Acar, and A Taylan Cemgil. Link prediction in heterogeneous data via generalized coupled tensor factorization. *Data Mining and Knowledge Discovery*, 29(1):203–236, 2015. 17, 18

[ED15] Evangelos E. Papalexakis and A. Seza Dougruöz. Understanding multilingual social networks in online immigrant communities. WWW Companion, 2015. 7

[EFS$^+$] Evangelos E. Papalexakis., Alona Fyshe, Nicholas D. Sidiropoulos, Partha Pratim Talukdar, Tom M. Mitchell, and Christos Faloutsos. Good-enough brain model: Challenges, algorithms and discoveries in multi-subject experiments. In *ACM KDD'14*. 5, 6, 163, 232

[EM13] Dóra Erdos and Pauli Miettinen. Walk'n'merge: A scalable algorithm for boolean tensor factorization. In *Data Mining (ICDM), 2013 IEEE 13th International Conference on*, pages 1037–1042. IEEE, 2013. 54, 56, 140

[ENR14] Enron e-mail dataset. `http://www.cs.cmu.edu/~enron/`, Last accessed: 9/9/2014. 72, 73

[EPL09] Nathan Eagle, Alex Sandy Pentland, and David Lazer. Inferring friendship network structure by using mobile phone data. *Proceedings of the National Academy of Sciences*, 106(36):15274–15278, 2009. 149, 197

[ESSF12] Eric Enge, Stephan Spencer, Jessie Stricchiola, and Rand Fishkin. *The art of SEO*. O'Reilly, 2012. 200

[Fed49] Federal Communications Commission. Editorializing by broadcast licensees. Washington, DC: GPO, 1949. 199

[FF94] Donald W Fausett and Charles T Fulton. Large least squares problems involving kronecker products. *SIAM Journal on Matrix Analysis and Applications*, 15(1):219–227, 1994. 134

[FFDM12] Alona Fyshe, Emily B Fox, David B Dunson, and Tom M Mitchell. Hierarchical latent dictionaries for models of brain activation. In *AISTATS*, pages 409–421, 2012. 166, 181

[FGG07] MS Fabian, K Gjergji, and W Gerhard. Yago: A core of semantic knowledge unifying wordnet and wikipedia. In *16th International World Wide Web Conference, WWW*, pages 697–706, 2007. 43

[FI11] Cédric Févotte and Jérôme Idier. Algorithms for nonnegative matrix factorization with the β-divergence. *Neural Computation*, 23(9):2421–2456, 2011. 142

[Fle71] Joseph L Fleiss. Measuring nominal scale agreement among many raters. *Psychological bulletin*, 76(5):378, 1971. 218

[FSF99] Angela D Friederici, Karsten Steinhauer, and Stefan Frisch. Lexical integration: Sequential effects of syntactic and semantic information. *Memory & Cognition*, 27(3):438–453, 1999. 182

[GC07] Zhiwei Guan and Edward Cutrell. An eye tracking study of the effect of target rank on web search. In *SIGCHI conference on Human factors in computing systems*, pages 417–420. ACM, 2007. 200

[GKRM03] Michael D Greicius, Ben Krasnow, Allan L Reiss, and Vinod Menon. Functional connectivity in the resting brain: a network analysis of the default mode hypothesis. *PNAS*, 100(1):253–258, 2003. 181

[GKT13] Lars Grasedyck, Daniel Kressner, and Christine Tobler. A literature survey of low-rank tensor approximation techniques. *GAMM-Mitteilungen*, 36(1):53–78, 2013. 21

[Gra10] Lars Grasedyck. Hierarchical singular value decomposition of tensors. *SIAM Journal on Matrix Analysis and Applications*, 31(4):2029–2054, 2010. 31

[GS05] Antonio Gulli and Alessio Signorini. Building an open source meta-search engine. In *14th international conference on World Wide Web*, pages 1004–1005. ACM, 2005. 203

[GW96] Susan Gauch and Guijun Wang. Information fusion with profusion. In *1st World Conference of the Web Society*, 1996. 201

[Har70] R.A. Harshman. Foundations of the parafac procedure: Models and conditions for an" explanatory" multimodal factor analysis. 1970. 14, 15, 110, 111, 130, 131, 140, 192, 205

[Har72a] R.A. Harshman. Determination and proof of minimum uniqueness conditions for PARAFAC-1. *UCLA Working Papers in Phonetics*, 22:111–117, 1972. 110, 111

[Har72b] Richard A Harshman. Parafac2: Mathematical and technical notes. *UCLA working papers in phonetics*, 22(3044):122215, 1972. 36

[Har78] Richard A Harshman. Models for analysis of asymmetrical relationships among n objects or stimuli. In *First Joint Meeting of the Psychometric Society* *and the Society for Mathematical Psychology, McMaster University, Hamilton, Ontario*, 1978. 29

[Har84] Richard A Harshman. How can i know if it's real? *A catalog of diagnostics for use with three-mode factor analysis and multidimensional scaling*, pages 566–591, 1984. 37

[HDLL08] Heng Huang, Chris Ding, Dijun Luo, and Tao Li. Simultaneous tensor subspace selection and clustering: the equivalence of high order svd and k-means clustering. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge Discovery and Data mining*, pages 327–335. ACM, 2008. 15

[Hei95] Willem J Heiser. Convergent computation by iterative majorization: theory and applications in multidimensional data analysis. *Recent advances in descriptive multivariate analysis*, pages 157–189, 1995. 142

[HGS14] Joyce C Ho, Joydeep Ghosh, and Jimeng Sun. Marble: high-throughput phenotyping from electronic health records via sparse nonnegative tensor factorization. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 115–124. ACM, 2014. 48, 141

[HHL03] Richard A Harshman, Sungjin Hong, and Margaret E Lundy. Shifted factor analysisâĂŤpart i: Models and properties. *Journal of chemometrics*, 17(7):363–378, 2003. 44

[Hit27] Frank L Hitchcock. The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics*, 6(1):164–189, 1927. 14

[HK09] Wolfgang Hackbusch and Stefan Kühn. A new scheme for the tensor representation. *Journal of Fourier Analysis and Applications*, 15(5):706–722, 2009. 31

[HKP$^+$14] Lifang He, Xiangnan Kong, S Yu Philip, Ann B Ragin, Zhifeng Hao, and Xiaowei Yang. Dusk: A dual structure-preserving kernel for supervised tensor learning with applications to neuroimages. *matrix*, 3(1):2, 2014. 44, 140

[HL13] Christopher J Hillar and Lek-Heng Lim. Most tensor problems are np-hard. *Journal of the ACM (JACM)*, 60(6):45, 2013. 130, 140

[HM75] C. Hung and T.L. Markham. The moore-penrose inverse of a partitioned matrix m= adbc. *Linear Algebra and its Applications*, 11(1):73–86, 1975. 95

[HMA$^+$14] Furong Huang, Sergiy Matusevych, Anima Anandkumar, Nikos Karampatziakis, and Paul Mineiro. Distributed latent dirichlet allocation via tensor factorization. 2014. 43

[HNHA13] Furong Huang, UN Niranjan, Mohammad Umar Hakeem, and Animashree Anandkumar. Fast detection of overlapping communities via online tensor methods. *arXiv preprint arXiv:1309.0787*, 2013. 40, 43

[HP04] Gregory Hickok and David Poeppel. Dorsal and ventral streams: a framework for understanding aspects of the functional anatomy of language. *Cognition*, 92(1-2):67–99, 2004. 177

[HSH$^+$10] Jianbin Huang, Heli Sun, Jiawei Han, Hongbo Deng, Yizhou Sun, and Yaguang Liu. Shrink: a structural clustering algorithm for detecting hierarchical communities in networks. In *CIKM*, pages 219–228, 2010. 188

[HSMK$^+$13] Aniko Hannak, Piotr Sapiezynski, Arash Molavi Kakhki, Balachander Krishnamurthy, David Lazer, Alan Mislove, and Christo Wilson. Measuring personalization of web search. In *22nd international conference on World Wide Web*, pages 527–538. ACM, 2013. 202, 210

[HZJ$^+$06] Yong He, Yufeng Zang, Tianzi Jiang, Gaolang Gong, Sheng Xie, and Jiangxi Xiao. Handedness-related functional connectivity using low-frequency blood oxygenation level-dependent fluctuations. *Neuroreport*, 17(1):5–8, 2006. 178

[Jae01] Herbert Jaeger. The âĂIJecho stateâĂİ approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148:34, 2001. 173

[Jae02a] Herbert Jaeger. Adaptive nonlinear system identification with echo state networks. In *Advances in neural information processing systems*, pages 593–600, 2002. 174

[Jae02b] Herbert Jaeger. *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach*. GMD-Forschungszentrum Informationstechnik, 2002. 173

[Jae07] H. Jaeger. Echo state network. 2(9):2330, 2007. revision 138672. 173, 174

[JCW$^+$14] Meng Jiang, Peng Cui, Fei Wang, Xinran Xu, Wenwu Zhu, and Shiqiang Yang. Fema: flexible evolutionary multi-faceted analysis for dynamic behavioral pattern discovery. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1186–1195. ACM, 2014. 41, 139

[JJK16] ByungSoo Jeon, LSI Jeon, and U Kang. Scout: Scalable coupled matrix-tensor factorizationâĂŤalgorithm and discoveries. In *ICDE*. IEEE, 2016. 57

[JPKF15] Inah Jeon, Evangelos E Papalexakis, U Kang, and Christos Faloutsos. Haten2: billion-scale tensor decompositions. In *ICDE*, 2015. 42, 52, 56, 57, 58

[JS04] T. Jiang and N.D. Sidiropoulos. Kruskal's permutation lemma and the identification of CANDECOMP/PARAFAC and bilinear models with constant modulus constraints. *IEEE Transactions on Signal Processing*, 52(9):2625–2636, 2004. 15, 22, 111

[KABO10] Alexandros Karatzoglou, Xavier Amatriain, Linas Baltrunas, and Nuria Oliver. Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 79–86. ACM, 2010. 29, 45

[KB54] G. R. Klare and B. Buck. *Know Your Reader: The scientific approach to readability*. Heritage House, 1954. 220

[KB06] T.G. Kolda and B.W. Bader. The tophits model for higher-order web link analysis. In *Workshop on Link Analysis, Counterterrorism and Security*, volume 7, pages 26–29, 2006. 130

[KB09] T.G. Kolda and B.W. Bader. Tensor decompositions and applications. *SIAM review*, 51(3), 2009. 13, 15, 21, 22, 28, 30, 100, 192, 194, 205

[KBK05] Tamara G Kolda, Brett W Bader, and Joseph P Kenny. Higher-order web link analysis using multilinear algebra. In *Data Mining, Fifth IEEE International Conference on*, pages 8–pp. IEEE, 2005. 41, 51, 56, 139, 140

[KC12] Mijung Kim and K Selccuk Candan. Decomposition-by-normalization (DBN): leveraging approximate functional dependencies for efficient tensor decomposition. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 355–364. ACM, 2012. 55, 56

[KDL80] Pieter M Kroonenberg and Jan De Leeuw. Principal component analysis of three-mode data by means of alternating least squares algorithms. *Psychometrika*, 45(1):69–97, 1980. 132

[Kie93] Henk AL Kiers. An alternating least squares algorithm for parafac2 and three-way dedicom. *Computational Statistics & Data Analysis*, 16(1):103–118, 1993. 30, 37

[Kie00] H.A.L. Kiers. Towards a standardized notation and terminology in multiway analysis. *Journal of Chemometrics*, 14(3):105–122, 2000. 87

[KK03] Henk AL Kiers and Albert Kinderen. A fast method for choosing the numbers of components in tucker3 analysis. *British Journal of Mathematical and Statistical Psychology*, 56(1):119–125, 2003. 39, 153

[KLPM10] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is twitter, a social network or a news media? In *19th international conference on World wide web*, pages 591–600. ACM, 2010. 203

[KPHF12] U Kang, Evangelos Papalexakis, Abhay Harpale, and Christos Faloutsos. Gigatensor: scaling tensor analysis up by 100 times-algorithms and discoveries. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 316–324. ACM, 2012. 4, 42, 52, 56, 57, 130, 140, 204

[KR02] Eleftherios Kofidis and Phillip A Regalia. On the best rank-1 approximation of higher-order supersymmetric tensors. *SIAM Journal on Matrix Analysis and Applications*, 23(3):863–884, 2002. 51

[Kro08] P.M. Kroonenberg. *Applied multiway data analysis*. Wiley, 2008. 112

[Kru77] J.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. 15, 22, 111

[KS08] Tamara G Kolda and Jimeng Sun. Scalable tensor decompositions for multi-aspect data mining. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 363–372. IEEE, 2008. 40, 52, 55, 57, 58, 79, 100, 107, 130, 139, 140

[KSM15] Alexander Kott, Ananthram Swami, and Patrick McDaniel. Six potential game-changers in cyber security: Towards priorities in cyber science and engineering. *arXiv preprint arXiv:1511.00509*, 2015. 4, 233

[KT12] Daniel Kressner and Christine Tobler. htucker–a matlab toolbox for tensors in hierarchical tucker format. *Mathicse, EPF Lausanne*, 2012. 31, 32

[Kuh55] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955. 68, 91, 92

[LCY$^+$10] Tao Lei, Rui Cai, Jiang-Ming Yang, Yan Ke, Xiaodong Fan, and Lei Zhang. A pattern tree-based approach to learning URL normalization rules. In *19th international conference on World Wide Web*, pages 611–620. ACM, 2010. 202, 210

[Lew12] Dirk Lewandowski. *Web search engine research*. Emerald Group Publishing, 2012. 201, 202

[LF73] F. W. Lancaster and E. G. Fayen. *Information Retrieval On-Line*. Melville Publishing Co., 1973. 201

[LG98] Steve Lawrence and C Lee Giles. Searching the world wide web. *Science*, 280(5360):98–100, 1998. 201

[Lju99] Lennart Ljung. *System identification*. Wiley Online Library, 1999. 174, 182

[LK14] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014. 149

[LKH05] Sang Ho Lee, Sung Jin Kim, and Seok Hoo Hong. On URL normalization. In *Computational Science and Its Applications–ICCSA 2005*, pages 1076–1085. Springer, 2005. 202, 210

[LLM10] Jure Leskovec, Kevin J. Lang, and Michael W. Mahoney. Empirical comparison of algorithms for network community detection. In *WWW*, pages 631–640, 2010. 188

[LMWY13] Ji Liu, Przemyslaw Musialski, Peter Wonka, and Jieping Ye. Tensor completion for estimating missing values in visual data. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(1):208–220, 2013. 49

[LNK10] Hady W Lauw, Alexandros Ntoulas, and Krishnaram Kenthapadi. Estimating the quality of postings in the real-time web. In *Proc. of SSM conference*, 2010. 204

[Loa00] Charles F Van Loan. The ubiquitous kronecker product. *Journal of computational and applied mathematics*, 123(1):85–100, 2000. 133

[LPV11] 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. 21

[LS99] Daniel D Lee and H Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999. 25

[LS15] Athanasios P Liavas and Nicholas D Sidiropoulos. Parallel algorithms for constrained tensor factorization via alternating direction method of multipliers. *Signal Processing, IEEE Transactions on*, 63(20):5450–5463, 2015. 54, 56

[LSC$^+$09] Yu-Ru Lin, Jimeng Sun, Paul Castro, Ravi Konuru, Hari Sundaram, and Aisling Kelliher. Metafac: community discovery via relational hypergraph factorization. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 527–536. ACM, 2009. 17, 26, 40, 86, 139

[LT08] S. Liu and G. Trenkler. Hadamard, khatri-rao, kronecker and other matrix products. *International Journal of Information and Systems Sciences*, pages 160–177, 2008. 94

[MDMT11] Yang Mu, Wei Ding, Melissa Morabito, and Dacheng Tao. Empirical discriminative tensor analysis for crime forecasting. In *Knowledge Science, Engineering and Management*, pages 293–304. Springer, 2011. 47

[MGC10] Mari-Carmen Marcos and Cristina González-Caro. Comportamiento de los usuarios en la página de resultados de los buscadores. un estudio basado en eye tracking. *El profesional de la información*, 19(4):348–358, 2010. 210

[MGD$^+$09] Vincenzo Maltese, Fausto Giunchiglia, Kerstin Denecke, Paul Lewis, Cornelia Wallner, Anthony Baldry, and Devika Madalli. *On the interdisciplinary foundations of diversity*. University of Trento, 2009. 200

[MGF11] K. Maruhashi, F. Guo, and C. Faloutsos. Multiaspectforensics: Pattern mining on large-scale heterogeneous networks with tensor analysis. In *Proceedings of the Third International Conference on Advances in Social Network Analysis and Mining*, 2011. 49, 130

[MH09] Morten Mørup and Lars Kai Hansen. Automatic relevance determination for multi-way models. *Journal of Chemometrics*, 23(7-8):352–363, 2009. 39, 148, 153

[MHA$^+$08] Morten Mørup, Lars Kai Hansen, Sidse Marie Arnfred, Lek-Heng Lim, and Kristoffer Hougaard Madsen. Shift-invariant multilinear decomposition of neuroimaging data. *NeuroImage*, 42(4):1439–1450, 2008. 44, 162, 236

[MHM11] Morten Mørup, Lars Kai Hansen, and Kristoffer Hougaard Madsen. Modeling latency and shape changes in trial based neuroimaging data. In *Signals, Systems and Computers (ASILOMAR), 2011 Conference Record of the Forty Fifth Asilomar Conference on*, pages 439–443. IEEE, 2011. 44, 236

[Mie11] Pauli Miettinen. Boolean tensor factorizations. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pages 447–456. IEEE, 2011. 26, 29

[MM15] Saskia Metzler and Pauli Miettinen. Clustering Boolean tensors. *Data Mining and Knowledge Discovery 29(5)*, page 1343âĂŞ1373, 2015. 39, 153

[MMD06] M.W. Mahoney, M. Maggioni, and P. Drineas. Tensor-cur decompositions for tensor-based data. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 327–336. ACM, 2006. 62

[MMD08] Michael W Mahoney, Mauro Maggioni, and Petros Drineas. Tensor-cur decompositions for tensor-based data. *SIAM Journal on Matrix Analysis and Applications*, 30(3):957–987, 2008. 57

[MRA13] Juan Martinez-Romo and Lourdes Araujo. Detecting malicious tweets in trending topics using a statistical analysis of language. *Expert Systems with Applications*, 40(8):2992–3000, 2013. 203

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. 196

[MSC$^+$08] T.M. Mitchell, S.V. Shinkareva, A. Carlson, K.M. Chang, V.L. Malave, R.A. Mason, and M.A. Just. Predicting human brain activity associated with the meanings of nouns. *Science*, 320(5880):1191–1195, 2008. 158

[MTM12] Brian Murphy, Partha Talukdar, and Tom Mitchell. Selecting corpus-semantic models for neurolinguistic decoding. In *First Joint Conference on Lexical and Computational Semantics (* SEM)*, pages 114–123, 2012. 158

[MWP$^+$14] Ching-Hao Mao, Chung-Jung Wu, Evangelos E Papalexakis, Christos Faloutsos, and Tien-Cheu Kao. MalSpot: Multi2 malicious network behavior patterns analysis. In *PAKDD 2014*, 2014. 7, 8, 49, 233

[MYL02] Weiyi Meng, Clement Yu, and King-Lup Liu. Building efficient and effective metasearch engines. *ACM Computing Surveys*, 34(1):48–89, 2002. 203

[Neu69] H Neudecker. A note on kronecker matrix products and matrix equation systems. *SIAM Journal on Applied Mathematics*, 17(3):603–606, 1969. 133

[NHF12] Kyosuke Nishida, Takahide Hoshide, and Ko Fujimura. Improving tweet stream classification by detecting changes in word probability. In *35th international ACM SIGIR conference on Research and development in information retrieval)*, pages 971–980. ACM, 2012. 209

[NHTK12] Atsuhiro Narita, Kohei Hayashi, Ryota Tomioka, and Hisashi Kashima. Tensor factorization using auxiliary information. *Data Mining and Knowledge Discovery*, 25(2):298–324, 2012. 17

[NMSP10] Dimitri Nion, Kleanthis N Mokios, Nicholas D Sidiropoulos, and Alexandros Potamianos. Batch and adaptive parafac-based blind separation of convolutive speech mixtures. *Audio, Speech, and Language Processing, IEEE Transactions on*, 18(6):1193–1207, 2010. 49

[NS09] Dimitri Nion and Nicholas D Sidiropoulos. Adaptive algorithms to track the parafac decomposition of a third-order tensor. *Signal Processing, IEEE Transactions on*, 57(6):2299–2310, 2009. 27, 108

[NTK11] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 809–816, 2011. 30

[NTK12] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. Factorizing yago: scalable machine learning for linked data. In *Proceedings of the 21st international conference on World Wide Web*, pages 271–280. ACM, 2012. 30, 43

[Ose11] Ivan V Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011. 33, 34

[PAB+05] R. Pang, M. Allman, M. Bennett, J. Lee, V. Paxson, and B. Tierney. A first look at modern enterprise traffic. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, pages 2–2. USENIX Association, 2005. 72

[PAI13] Evangelos E Papalexakis, Leman Akoglu, and Dino Ienco. Do more views of a graph help? community detection and clustering in multi-graphs. In *Information Fusion (FUSION), 2013 16th International Conference on*, pages 899–905. IEEE, 2013. 8, 41, 130, 149, 187

[Pap16] Evangelos E Papalexakis. Automatic unsupervised tensor mining with quality assessment. In *SIAM SDM*, 2016. 5, 139, 232

[PB98] Martin J Pickering and Holly P Branigan. The representation of verbs: Evidence from syntactic priming in language production. *Journal of Memory and Language*, 39(4):633–651, 1998. 182

[PBR12] Kristin Purcell, Joanna Brenner, and Lee Rainie. *Search engine use 2012*. Pew Internet & American Life Project, 2012. 199

[PC09] A.H. Phan and A. Cichocki. Block decomposition for very large-scale nonnegative tensor factorization. In *Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP), 2009 3rd IEEE International Workshop on*, pages 316–319. IEEE, 2009. 53, 54, 56

[PC11] A.H. Phan and A. Cichocki. Parafac algorithms for large-scale problems. *Neurocomputing*, 74(11):1970–1984, 2011. 108

[PCVS15] Ioakeim Perros, Robert Chen, Richard Vuduc, and Jimeng Sun. Sparse hierarchical tucker factorization and its application to healthcare. In *Data Mining (ICDM), 2015 IEEE 15th International Conference on*. IEEE, 2015. 32, 48, 57, 58, 236

[PF15] E.E. Papalexakis and C. Faloutsos. Fast efficient and scalable core consistency diagnostic for the parafac decomposition for big sparse tensors. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 2015. 39, 129, 147, 149, 153, 233

[PFS] E.E. Papalexakis, C. Faloutsos, and N.D. Sidiropoulos. Tensors for data mining and data fusion: Models, applications, and scalable algorithms. *ACM Trans. on Intelligent Systems and Technology*. 11, 21

[PFS12] Evangelos E Papalexakis, Christos Faloutsos, and Nicholas D Sidiropoulos. ParCube: Sparse parallelizable tensor decompositions. In *Machine Learning and Knowledge Discovery in Databases*, pages 521–536. Springer, 2012. 2, 4, 7, 40, 42, 49, 53, 54, 56, 61, 62, 130, 140, 232

[PFS+14a] Evangelos E Papalexakis, Alona Fyshe, Nicholas D Sidiropoulos, Partha Pratim Talukdar, Tom M Mitchell, and Christos Faloutsos. Good-enough brain model: challenges, algorithms and discoveries in multi-subject experiments. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 95–104. ACM, 2014. 166, 167, 170, 174, 176

[PFS+14b] Evangelos E Papalexakis, Alona Fyshe, Nicholas D Sidiropoulos, Partha Pratim Talukdar, Tom M Mitchell, and Christos Faloutsos. Good-enough brain model: Challenges, algorithms, and discoveries in multisubject experiments. *Big Data*, 2(4):216–229, 2014. 163

[PFS15] Evangelos E Papalexakis, Christos Faloutsos, and Nicholas D Sidiropoulos. Parcube: Sparse parallelizable candecomp-parafac tensor decomposition. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 10(1):3, 2015. 61

[Pir09] Ari Pirkola. The effectiveness of web search engines to index new sites from different countries. *Information Research: An International Electronic Journal*, 14(2), 2009. 202

[PK15] Evangelia Pantraki and Constantine Kotropoulos. Automatic image tagging and recommendation via parafac2. In *Machine Learning for Signal Processing (MLSP), 2015 IEEE 25th International Workshop on*, pages 1–6. IEEE, 2015. 46

[PMS+14] Evangelos E Papalexakis, Tom M Mitchell, Nicholas D Sidiropoulos, Christos Faloutsos, Partha Pratim Talukdar, and Brian Murphy. Turbo-smt: Accelerating coupled sparse matrix-tensor factorizations by 200x. In *SIAM SDM*, 2014. 3, 5, 6, 45, 53, 85, 157, 232

[PMS+16] Evangelos E Papalexakis, Tom M Mitchell, Nicholas D Sidiropoulos, Christos Faloutsos, Partha Pratim Talukdar, and Brian Murphy. Turbo-smt: Parallel coupled sparse matrix-tensor factorizations and applications. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 2016. 5, 85, 157, 232

[PPF15] Evangelos E Papalexakis, Konstantinos Pelechrinis, and Christos Faloutsos. Location based social network analysis using tensors and signal processing tools. *IEEE CAMSAP*, 15, 2015. 7

[PPHM09] Mark Palatucci, Dean Pomerleau, Geoffrey Hinton, and Tom Mitchell. Zero-shot learning with semantic output codes. *Advances in neural information processing systems*, 22:1410–1418, 2009. 158

[PS11] E.E. Papalexakis and N.D. Sidiropoulos. Co-clustering as multilinear decomposition with sparse latent factors. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 2064–2067. IEEE, 2011. 63, 73, 192

[PSB13] Evangelos E Papalexakis, Nicholas D Sidiropoulos, and Rasmus Bro. From k-means to higher-way co-clustering: multilinear decomposition with sparse latent factors. *Signal Processing, IEEE Transactions on*, 61(2):493–506, 2013. 3, 26, 40, 51, 63, 71, 72, 73, 80, 96, 108, 133

[PZZW10] Jing Peng, Daniel Dajun Zeng, Huimin Zhao, and Fei-yue Wang. Collaborative filtering in social tagging systems based on joint item-tag recommendations. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 809–818. ACM, 2010. 46

[RAB$^+$09] Catharine H Rankin, Thomas Abrams, Robert J Barry, Seema Bhatnagar, David F Clayton, John Colombo, Gianluca Coppola, Mark A Geyer, David L Glanzman, Stephen Marsland, et al. Habituation revisited: an updated and revised description of the behavioral characteristics of habituation. *Neurobiology of learning and memory*, 92(2):135–138, 2009. 181

[Ren10] Steffen Rendle. Factorization machines. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 995–1000. IEEE, 2010. 46

[RGAH11] Daniel M Romero, Wojciech Galuba, Sitaram Asur, and Bernardo A Huberman. Influence and passivity in social media. In *Machine learning and knowledge discovery in databases*, pages 18–33. Springer, 2011. 203

[RHMF12] Md Sazzadur Rahman, Ting-Kai Huang, Harsha V Madhyastha, and Michalis Faloutsos. Efficient and scalable socware detection in online social networks. In *USENIX Security Symposium*, pages 663–678, 2012. 203

[RHS10] Tom Rowlands, David Hawking, and Ramesh Sankaranarayana. New-web search with microblog annotations. In *19th international conference on World wide web*, pages 1293–1296. ACM, 2010. 205

[Ris83] J. Rissanen. A universal prior for integers and estimation by minimum description length. *The Annals of Statistics*, 11(2):416–431, 1983. 190, 191

[RP11] Matthew Rocklin and Ali Pinar. Latent clustering on graphs with multiple edge types. In *WAW*, pages 38–49, 2011. 189

[RSSK14] Niranjay Ravindran, Nicholas D Sidiropoulos, Shaden Smith, and George Karypis. Memory-efficient parallel computation of tensor and matrix products for big tensor decomposition. In *Signals, Systems and Computers, 2014 48th Asilomar Conference on*, pages 581–585. IEEE, 2014. 53, 54, 56

[RST10] Steffen Rendle and Lars Schmidt-Thieme. Pairwise interaction tensor factorization for personalized tag recommendation. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 81–90. ACM, 2010. 45, 46, 140

[RTW16] Read the web. `http://rtw.ml.cmu.edu/rtw/`, Last accessed: 2/13/2016. 42, 72

[Sak11] Vangelis Sakkalis. Review of advanced techniques for the estimation of brain connectivity measured with eeg/meg. *Computers in biology and medicine*, 41(12):1110–1117, 2011. 181

[SB00] N.D. Sidiropoulos and R. Bro. On the uniqueness of multilinear decomposition of N-way arrays. *Journal of chemometrics*, 14(3):229–239, 2000. 15, 22, 111

[SBGW04] A.K. Smilde, R. Bro, P. Geladi, and J. Wiley. *Multi-way analysis with applications in the chemical sciences*. Wiley, 2004. 112

[Sch14] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *CoRR*, abs/1404.7828, 2014. 182

[SE95] Erik Selberg and Oren Etzioni. Multi-service search and comparison using the metacrawler. In *4th international conference on World Wide Web*, 1995. 201

[SFD$^+$10] Bharath Sriram, Dave Fuhry, Engin Demir, Hakan Ferhatosmanoglu, and Murat Demirbas. Short text classification in twitter to improve information filtering. In *33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 841–842. ACM, 2010. 209

[SFPY07] Jimeng Sun, Christos Faloutsos, Spiros Papadimitriou, and Philip S. Yu. Graphscope: parameter-free mining of large time-evolving graphs. In *KDD*, 2007. 192

[SG08] Ajit P Singh and Geoffrey J Gordon. Relational learning via collective matrix factorization. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 650–658. ACM, 2008. 17

[SGL07] Ioannis D Schizas, Georgios B Giannakis, and Zhi-Quan Luo. Distributed estimation using reduced-dimensionality sensor observations. *IEEE TSP*, 55(8):4284–4299, 2007. 167

[SH05] Amnon Shashua and Tamir Hazan. Non-negative tensor factorization with applications to statistics and computer vision. In *Proceedings of the 22nd international conference on Machine learning*, pages 792–799. ACM, 2005. 25

[SHY$^+$11] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S Yu, and Tianyi Wu. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *VLDB*, 2011. 236

[SJBK06] Amanda Spink, Bernard J Jansen, Chris Blakely, and Sherry Koshman. A study of results overlap and uniqueness among major web search engines. *Information Processing & Management*, 42(5):1379–1391, 2006. 202

[SJW08] Amanda Spink, Bernard J Jansen, and Changru Wang. Comparison of major web search engine overlap: 2005 and 2007. In *14th Australasian World Wide Web Conference*, 2008. 202

[SK12] N.D. Sidiropoulos and A. Kyrillidis. Multi-way compressed sensing for sparse low-rank tensors. *IEEE Signal Processing Letters*, 19(11):757–760, 2012. 54, 108, 112, 113, 116, 122

[SK14] Kijung Shin and U Kang. Distributed methods for high-dimensional and large-scale tensor factorization. In *Data Mining (ICDM), 2014 IEEE International Conference on*, pages 989–994. IEEE, 2014. 55, 56

[SKB12] Adam Sadilek, Henry Kautz, and Jeffrey P Bigham. Finding your friends and following them to where you are. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 723–732. ACM, 2012. 164

[SM08] Ruslan Salakhutdinov and Andriy Mnih. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *Proceedings of the 25th international conference on Machine learning*, pages 880–887. ACM, 2008. 45

[SM12a] Motoki Shiga and Hiroshi Mamitsuka. A variational bayesian framework for clustering with multiple graphs. *IEEE Trans. Knowl. Data Eng.*, 24(4):577–590, 2012. 189

[SM12b] Natalie Jomini Stroud and Ashley Muddiman. Exposure to news and diverse views in the internet age. *ISJLP*, 8:605, 2012. 199

[SMML$^+$14] Igor Santos, Igor Miñambres-Marcos, Carlos Laorden, Patxi Galán-García, Aitor Santamaría-Ibirika, and Pablo García Bringas. Twitter content-based spam filtering. In *International Joint Conference SOCO'13-CISIS'13-ICEUTE'13*, pages 449–458. Springer, 2014. 203

[SPBW15] Aaron Schein, John Paisley, David M Blei, and Hanna Wallach. Bayesian poisson tensor factorization for inferring multilateral relations from sparse dyadic event counts. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1045–1054. ACM, 2015. 41

[SPF14] N Sidiropoulos, E Papalexakis, and C Faloutsos. Parallel randomly compressed cubes: A scalable distributed architecture for big tensor decomposition. *Signal Processing Magazine, IEEE*, 31(5):57–70, 2014. 5, 54, 56, 107, 116, 232

[SPL$^+$09] Liang Sun, Rinkal Patel, Jun Liu, Kewei Chen, Teresa Wu, Jing Li, Eric Reiman, and Jieping Ye. Mining brain region connectivity for alzheimer's disease study via sparse inverse covariance estimation. In *ACM SIGKDD*, pages 1335–1344. ACM, 2009. 173, 181

[SPP+12] G. Sudre, D. Pomerleau, M. Palatucci, L. Wehbe, A. Fyshe, R. Salmelin, and T. Mitchell. Tracking neural coding of perceptual and semantic features of concrete nouns. *NeuroImage*, 2012. 166, 175

[SRSK15] Shaden Smith, Niranjay Ravindran, Nicholas D Sidiropoulos, and George Karypis. Splatt: Efficient and parallel sparse tensor-matrix multiplication. In *29th IEEE International Parallel & Distributed Processing Symposium*, 2015. 53, 56

[SS07] A. Stegeman and N.D. Sidiropoulos. On Kruskal's uniqueness condition for the CANDECOMP/PARAFAC decomposition. *Linear Algebra and its Applications*, 420(2-3):540–552, 2007. 111

[StBDL06] A. Stegeman, J.M.F. ten Berge, and L. De Lathauwer. Sufficient conditions for uniqueness in CANDECOMP/PARAFAC and INDSCAL with random component matrices. *Psychometrika*, 71(2):219–229, 2006. 15, 22, 111

[STF06] J. Sun, D. Tao, and C. Faloutsos. Beyond streams and graphs: dynamic tensor analysis. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 374–383. ACM, 2006. 29, 140

[STH+08] Jimeng Sun, Charalampos E Tsourakakis, Evan Hoke, Christos Faloutsos, and Tina Eliassi-Rad. Two heads better than one: pattern discovery in time-evolving multi-aspect data. *Data Mining and Knowledge Discovery*, 17(1):111–128, 2008. 29, 236

[SWB00] A.K. Smilde, J.A. Westerhuis, and R. Boque. Multiway multiblock component and covariates regression models. *Journal of Chemometrics*, 14(3):301–331, 2000. 17

[SZL+05] J.T. Sun, H.J. Zeng, H. Liu, Y. Lu, and Z. Chen. Cubesvd: a novel approach to personalized web search. In *Proceedings of the 14th international conference on World Wide Web*, pages 382–390. ACM, 2005. 42

[TAHH12] Ke Tao, Fabian Abel, Claudia Hauff, and Geert-Jan Houben. Twinder: a search engine for twitter streams. In *Web Engineering*, pages 153–168. Springer, 2012. 203

[TB05] Giorgio Tomasi and Rasmus Bro. Parafac and missing values. *Chemometrics and Intelligent Laboratory Systems*, 75(2):163–180, 2005. 25, 95, 96

[TB06] Giorgio Tomasi and Rasmus Bro. A comparison of algorithms for fitting the parafac model. *Computational Statistics & Data Analysis*, 50(7):1700–1734, 2006. 22

[tBS02] Jos MF ten Berge and Nikolaos D Sidiropoulos. On uniqueness in candecomp/parafac. *Psychometrika*, 67(3):399–409, 2002. 15

[TK00] Marieke E Timmerman and Henk AL Kiers. Three-mode principal components analysis: Choosing the numbers of components and sensitivity to local optima. *British journal of mathematical and statistical psychology*, 53(1):1–16, 2000. 39

[TLD09] Wei Tang, Zhengdong Lu, and Inderjit S. Dhillon. Clustering with multiple graphs. In *ICDM*, pages 1016–1021, 2009. 188, 189, 194, 196, 198

[TRM11] Jaime Teevan, Daniel Ramage, and Merredith Ringel Morris. # twittersearch: a comparison of microblog search and web search. In *Fourth ACM international conference on Web search and data mining*, pages 35–44. ACM, 2011. 201, 204, 211, 217

[TS66] Richard F Thompson and William A Spencer. Habituation: a model phenomenon for the study of neuronal substrates of behavior. *Psychological review*, 73(1):16, 1966. 181

[TSL$^+$08] Dacheng Tao, Mingli Song, Xuelong Li, Jialie Shen, Jimeng Sun, Xindong Wu, Christos Faloutsos, and Stephen J Maybank. Bayesian tensor approach for 3-d face modeling. *Circuits and Systems for Video Technology, IEEE Transactions on*, 18(10):1397–1410, 2008. 50

[Tso10] Charalampos E Tsourakakis. Mach: Fast randomized tensor decompositions. In *SDM*, pages 689–700. SIAM, 2010. 57, 58

[TT12] Hikaru Takemura and Keishi Tajima. Tweet classification based on their lifetime duration. In *21st ACM international conference on Information and knowledge management*, pages 2367–2370. ACM, 2012. 209

[Tuc66] L.R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966. 15, 30

[Tur11] *Amazon Mechanical Turk, Requester Best Practices Guide*. Amazon Web Services, June 2011. 218

[TV12] Dardo Tomasi and Nora D Volkow. Resting functional connectivity of language networks: characterization and reproducibility. *Molecular psychiatry*, 17(8):841–854, 2012. 178

[TWL12] Lei Tang, Xufei Wang, and Huan Liu. Community detection via heterogeneous interaction analysis. *Data Min. Knowl. Discov.*, 25(1):1–33, 2012. 188, 189

[UC11] Ibrahim Uysal and W Bruce Croft. User oriented tweet ranking: a filtering approach to microblogs. In *20th ACM international conference on Information and knowledge management*, pages 2261–2264. ACM, 2011. 203

[Ver94] Michel Verhaegen. Identification of the deterministic part of mimo state space models given in innovations form from input-output data. *Automatica*, 30(1):61–74, 1994. 174

[vL07] Ulrike von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007. 188, 196

[VMCG09] Bimal Viswanath, Alan Mislove, Meeyoung Cha, and Krishna P. Gummadi. On the evolution of user interaction in facebook. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Social Networks (WOSN'09)*, August 2009. 72, 81, 149

[VSSBLC$^+$05] Pedro A Valdés-Sosa, Jose M Sánchez-Bornot, Agustín Lage-Castellanos, Mayrim Vega-Hernández, Jorge Bosch-Bayard, Lester Melie-García, and Erick Canales-Rodríguez. Estimating brain functional connectivity with sparse multivariate autoregression. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 360(1457):969–981, 2005. 181

[VT02] M. Vasilescu and D. Terzopoulos. Multilinear analysis of image ensembles: Tensorfaces. *Computer Vision ECCV 2002*, pages 447–460, 2002. 49

[VV07] Michel Verhaegen and Vincent Verdult. *Filtering and system identification: a least squares approach*. Cambridge university press, 2007. 182

[VVA$^+$06] Olivier Verscheure, Michail Vlachos, Aris Anagnostopoulos, Pascal Frossard, Eric Bouillet, and Philip S Yu. Finding "who is talking to whom" in voip networks via progressive stream clustering. In *IEEE ICDM*, pages 667–677. IEEE, 2006. 164, 182

[WCVM09] T. Wilderjans, E. Ceulemans, and I. Van Mechelen. Simultaneous analysis of coupled data blocks differing in size: A comparison of two weighting schemes. *Computational Statistics & Data Analysis*, 53(4):1086–1098, 2009. 18, 86, 87

[WD09] Ryen W White and Susan T Dumais. Characterizing and predicting search engine switching behavior. In *18th ACM conference on Information and knowledge management*, pages 87–96. ACM, 2009. 202

[Web14] William M Webberley. *Inferring Interestingness in Online Social Networks*. PhD thesis, Cardiff University, 2014. 203, 204

[WH88] Robert W Williams and Karl Herrup. The control of neuron number. *Annual review of neuroscience*, 11(1):423–453, 1988. 164

[WT13] David Wilkinson and Mike Thelwall. Search markets and search results: The case of Bing. *Library & Information Science Research*, 35(4):318–325, 2013. 202

[WZX14] Yilun Wang, Yu Zheng, and Yexiang Xue. Travel time estimation of a path using sparse trajectories. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 25–34, New York, NY, USA, 2014. ACM. 17, 47, 149, 152

[XCH+10] Liang Xiong, Xi Chen, Tzu-Kuo Huang, Jeff G Schneider, and Jaime G Carbonell. Temporal collaborative filtering with bayesian probabilistic tensor factorization. In *SDM*, volume 10, pages 211–222. SIAM, 2010. 45

[XKW+12] Zhiqiang Xu, Yiping Ke, Yi Wang, Hong Cheng, and James Cheng. A model-based approach to attributed graph clustering. In *SIGMOD*, pages 505–516, 2012. 188

[XYFS07] Xiaowei Xu, Nurcan Yuruk, Zhidan Feng, and Thomas A. J. Schweiger. Scan: a structural clustering algorithm for networks. In *KDD*, pages 824–833, 2007. 188

[YCY12] Tatsuya Yokota, Andrzej Cichocki, and Yukihiko Yamashita. Linked parafac/cp tensor decomposition and its fast implementation for multi-block tensor analysis. In *Neural Information Processing*, pages 84–91. Springer, 2012. 17

[YLLR12] Min-Chul Yang, Jung-Tae Lee, Seung-Wook Lee, and Hae-Chang Rim. Finding interesting posts in twitter based on retweet graph analysis. In *35th international ACM SIGIR conference on Research and development in information retrieval*, pages 1073–1074. ACM, 2012. 203

[YR14] Min-Chul Yang and Hae-Chang Rim. Identifying interesting twitter contents using topical analysis. *Expert Systems with Applications*, 41(9):4330–4336, 2014. 204

[YZXS11] Jing Yuan, Yu Zheng, Xing Xie, and Guangzhong Sun. Driving with knowledge from the physical world. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 316–324. ACM, 2011. 149

[ZB07] Dengyong Zhou and Christopher J. C. Burges. Spectral clustering and transductive learning with multiple views. In *ICML*, pages 1159–1166, 2007. 189

[ZCZ+10] Vincent Wenchen Zheng, Bin Cao, Yu Zheng, Xing Xie, and Qiang Yang. Collaborative filtering meets mobile recommendation: A user-centered approach. In *AAAI*, volume 10, pages 236–241, 2010. 17, 46

[ZLW+14] Yu Zheng, Tong Liu, Yilun Wang, Yanmin Zhu, Yanchi Liu, and Eric Chang. Diagnosing new york city's noises with ubiquitous data. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 715–725. ACM, 2014. 47

[ZYW+15] Fuzheng Zhang, Nicholas Jing Yuan, David Wilkie, Yu Zheng, and Xing Xie. Sensing the pulse of urban refueling behavior: A perspective from taxi mobility. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 6(3):37, 2015. 47

[ZZC15] Q Zhao, L Zhang, and A Cichocki. Bayesian CP factorization of incomplete tensors with automatic rank determination. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 37:1751 – 1763, 2015. 39, 148, 153

[ZZXY12] Vincent W Zheng, Yu Zheng, Xing Xie, and Qiang Yang. Towards mobile intelligence: Learning from gps history data for collaborative recommendation. *Artificial Intelligence*, 184:17–37, 2012. 46