Large Scale Tensor Decompositions: Algorithmic Developments and Applications

Evangelos Papalexakis[⋆∗], U Kang[†] , Christos Faloutsos[⋆], Nicholas Sidiropoulos[§], Abhay Harpale[⋆]
[⋆] Carnegie Mellon University, [†] KAIST, [§] University of Minnesota

Abstract

Tensor decompositions are increasingly gaining popularity in data science applications. Albeit extremely powerful tools, scalability to truly large datasets for such decomposition algorithms is still a challenging problem. In this paper, we provide an overview of recent algorithmic developments towards the direction of scaling tensor decompositions to big data. We present an exact Map/Reduce based algorithm, as well as an approximate, fully parallelizable algorithm that is sparsity promoting. In both cases, careful design and implementation is key, so that we achieve scalability and efficiency. We showcase the effectiveness of our methods, by providing a variety of real world applications - whose volume previously rendered their analysis very hard, if not impossible- where our algorithms were able to discover interesting patterns and anomalies.

1 Introduction

Tensors and tensor decompositions are powerful tools, and are increasingly gaining popularity in data analytics and mining. Despite their power and popularity, tensor decompositions prove very challenging when it comes to scalability towards big data. Tensors are, essentially, multidimensional generalizations of matrices; for instance, a two dimensional tensor is a plain matrix, and a three dimensional tensor is a cubic structure.

As an example, consider a knowledge base, such as the "Read the Web" project [1] at Carnegie Mellon University, which consists of (noun phrase, context, noun phrase) triplets, such as ("Obama", "is the president of", "USA"). Figure 1 demonstrates how we can formulate this data as a three mode tensor and how we may analyze it in latent concepts, each one representing an entire cluster of noun phrases and contexts.

Alternatively, consider a social network, such as Facebook, where users interact with each other, and post on each others' "Walls". Given this posting activity over time, we can formulate a tensor of who interacted with whom and when; subsequently, by decomposing the tensor into concepts, as in Fig. 1, we are able to identify cliques of friends, as well as anomalies.

In this paper, we present a brief overview of tensor decompositions and their applications in the social media context, geared towards scalability.

[∗]Funding was provided to the first three authors by the U.S. ARO and DARPA under Contract Number W911NF-11-C-0088, by DTRA under contract No. HDTRA1-10-1-0120, by ARL under Cooperative Agreement Number W911NF-09-2-0053, and by grant NSF IIS-1247489. N. Sidiropoulos was partially supported by ARO contract W911NF-11-1-0500 and by NSF IIS-1247632 Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding parties

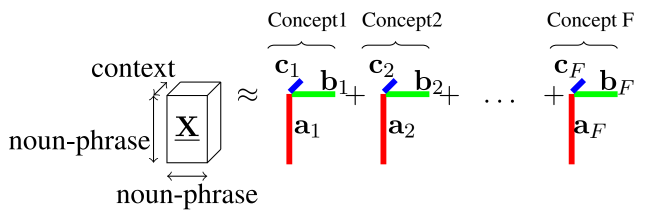


Figure 1: PARAFAC decomposition of three-way tensor as sum of F outer products (rank-one tensors), reminiscent of the rank-F Singular Value Decomposition of a matrix.

1.1 A note on notation

Tensors are denoted as $\mathcal{X}$. Matrices are denoted as $\mathbf{X}$. Vectors are denoted as $\mathbf{x}$. We use Matlab notation for matrix indexing, e.g. $\mathbf{A}(1,2)$ refers to the (1,2) element of $\mathbf{A}$, and $\mathbf{A}(:,1)$, refers to the entire first column of $\mathbf{A}$. The rest of the symbols used are defined throughout the text.

1.2 Tensors and the PARAFAC decomposition

Tensors are multidimensional matrices; in tensor terminology, each dimension is called a 'mode'. The most popular tensors are three mode ones, however, there exist applications that analyze tensors of higher dimensions. There is a rich literature on tensor decompositions; we refer the interested reader to [12] for a comprehensive overview thereof. This work focuses on the PARAFAC decomposition [8] (which is the one shown in Fig. 1).

The PARAFAC decomposition can be seen as a generalization of matrix factorizations, such as the Singular Value Decomposition, in higher dimensions, or as they are referred to in tensor literature, *modes*.

The PARAFAC [7, 8] (also known as CANDECOMP/PARAFAC or Canonical Polyadic Decomposition) tensor decomposition of $\mathcal{X}$ in F components is $\mathcal{X} \approx \sum_{f=1}^{F} \mathbf{a}_f \circ \mathbf{b}_f \circ \mathbf{c}_f$, where $[\mathbf{a} \circ \mathbf{b} \circ \mathbf{c}](i,j,k) = \mathbf{a}(i)\mathbf{b}(j)\mathbf{c}(k)$ and denotes the three mode outer product.

Often, we represent the PARAFAC decomposition as a triplet of matrices $\mathbf{A}, \mathbf{B}$, and $\mathbf{C}$, i.e. the f-th column of which contains $\mathbf{a}_f, \mathbf{b}_f$ and $\mathbf{c}_f$, respectively.

Definition 1 (Tensor Matricization): We may matricize a tensor $\mathcal{X} \in R^{I \times J \times K}$ in the following three ways: $\mathbf{X}_{(\mathbf{1})}$ of size $(I \times JK)$, $\mathbf{X}_{(\mathbf{2})}$ of size $(J \times IK)$ and $\mathbf{X}_{(\mathbf{3})}$ of size $(K \times IJ)$. We refer the interested reader to [10] for details.

Definition 2 (Kronecker product): The Kronecker product of $\mathbf{A}$ and $\mathbf{B}$ is:

$$\mathbf{A} \otimes \mathbf{B} := \begin{bmatrix} \mathbf{B}\mathbf{A}(1,1) & \cdots & \mathbf{B}\mathbf{A}(1,J_1) \\ \vdots & \ddots & \vdots \\ \mathbf{B}\mathbf{A}(I_1,1) & \cdots & \mathbf{B}\mathbf{A}(I_1,J_1) \end{bmatrix}$$

If $\mathbf{A}$ is of size $I_1 \times J_1$ and $\mathbf{B}$ of size $I_2 \times J_2$, then $\mathbf{A} \otimes \mathbf{B}$ is of size $I_1 I_2 \times J_1 J_2$.

Definition 3 (Khatri-Rao product): The Khatri-Rao product (or column-wise Kronecker product) $(\mathbf{A} \odot \mathbf{B})$, where $\mathbf{A}, \mathbf{B}$ have the same number of columns, say F, is defined as:

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

If $\mathbf{A}$ is of size $I \times F$ and $\mathbf{B}$ is of size $J \times F$ then $(\mathbf{A} \odot \mathbf{B})$ is of size $IJ \times F$.

The Alternating Least Squares Algorithm for PARAFAC. The most popular algorithm for fitting the PARAFAC decomposition is the Alternating Least Squares (ALS). The ALS algorithm consists of three steps, each one being a conditional update of one of the three factor matrices, given the other two. Without delving into details, the update of, e.g., the factor matrix $\mathbf{A}$, keeping $\mathbf{B}, \mathbf{C}$ fixed, involves the computation and pseudoinversion of $(\mathbf{C} \odot \mathbf{B})$ (and accordingly for the updates of $\mathbf{B}, \mathbf{C}$). For a detailed overview of the ALS algorithm, see [7, 8, 12].

2 Related Work

2.1 Applications

In [11], the authors incorporate contextual information to the traditional HITS algorithm, formulating it as a tensor decomposition. In [5] the authors analyze the ENRON email social network, formulating it as a tensor. In [2] the authors introduce a tensor-based framework in order to identify epileptic seizures. In [17], the authors use tensors in order to incorporate user click information and improve web search. The list continues, including applications such as [13], [16], and [2].

2.2 State of the art

The standard framework for working with tensors is Matlab; there exist two well known toolboxes, both of very high quality: The Tensor Toolbox for Matlab [4, 6] (specializing in sparse tensors) and the N-Way Toolbox for Matlab [3] (specializing in dense tensors).

In [15], the authors propose a partition-and-merge scheme for the PARAFAC decomposition which, however, does not offer factor sparsity. In terms of parallel algorithms, [19] introduces parallelization strategies for speeding up each factor matrix update step in the context of alternating optimization. Finally, [16, 18] propose randomized, sampling based tensor decompositions (however, the focus is on a different tensor model, the so called Tucker3).

The latest developments on scalable tensor decompositions are the works summarized in this paper: in [9], a massively distributed Map/Reduce version of PARAFAC is proposed, where, after careful design, issues fatal to scalability are effectively alleviated. In [14], a sampling based, parallel and sparsity promoting, approximate PARAFAC decomposition is proposed.

3 Scaling Tensor Decompositions Up

3.1 Main Challenge

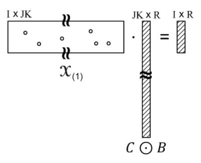
Previously, when describing the ALS algorithm, we mentioned that the update of $\mathbf{A}$ involves manipulation of $(\mathbf{C} \odot \mathbf{B})$. This is the very weakness of the traditional ALS algorithm, a naive implementation thereof will have to materialize matrices $(\mathbf{C} \odot \mathbf{B})$, $(\mathbf{C} \odot \mathbf{A})$, and $(\mathbf{B} \odot \mathbf{A})$, for the respective updates of $\mathbf{A}, \mathbf{B}$, and $\mathbf{C}$.

Problem 1 (Intermediate Data Explosion): The problem of having to materialize $(\mathbf{C} \odot \mathbf{B})$, $(\mathbf{C} \odot \mathbf{A})$, and $(\mathbf{B} \odot \mathbf{A})$ is defined as the intermediate data explosion.

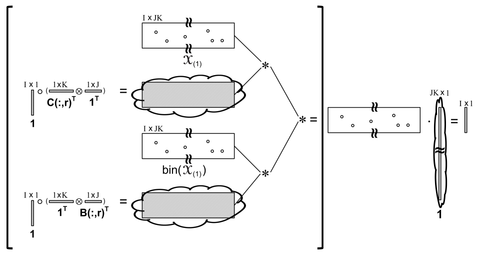
In order to give an idea of how devastating this intermediate data explosion problem is, consider the knowledge base dataset, such as the one referenced in the Introduction. The version of the data that we analyzed, consists of about $26 \cdot 10^6$ noun-phrases. Consequently, a naive implementation of ALS would generate and store a matrix of $\approx 7 \cdot 10^{14}$. As an indication of how devastating this choice is, we would probably need a data center's worth of storage, just to store this matrix, let alone manipulate it.

In [4], Bader et al. introduce a way to alleviate the above problem, when the tensor is stored in Matlab sparse format. However, this implementation is bound by Matlab's memory limitations.

In the following subsections we provide an overview of our two recent works, which both achieve scalability, pursuing two different directions.



(a)



(b)

Figure 2: Subfigure (a): The intermediate data explosion problem in computing $\mathbf{X_{(1)}}(\mathbf{C} \odot \mathbf{B})$. Although $\mathbf{X_{(1)}}$ is sparse, the matrix $\mathbf{C} \odot \mathbf{B}$ is very dense and long. Materializing $\mathbf{C} \odot \mathbf{B}$ requires too much storage: e.g., for $J = K \approx 26$ million as the main dataset of [9], $\mathbf{C} \odot \mathbf{B}$ explodes to *676 trillion* rows. Subfigure (b): Our solution to avoid the intermediate data explosion. The main idea is to decouple the two terms in the Khatri-Rao product, and perform algebraic operations using $\mathbf{X_{(1)}}$ and $\mathbf{C}$, and then $\mathbf{X_{(1)}}$ with $\mathbf{B}$, and combine the result. The symbols $\circ, \otimes, *$, and $\cdot$ represents the outer, Kronecker, Hadamard (element-wise), and the standard product, respectively. Shaded matrices are dense, and empty matrices with several circles are sparse. The clouds surrounding matrices represent that the matrices are ***not*** materialized. Note that the matrix $\mathbf{C} \odot \mathbf{B}$ is never constructed, and the largest dense matrix is either the $\mathbf{B}$ or the $\mathbf{C}$ matrix. Both figures are taken from [9].

3.2 GigaTensor

GigaTensor, which was introduced in [9], is a highly scalable, distributed implementation of the PARAFAC decomposition. Scalability was achieved through a crucial simplification of the algorithm, a series of careful design choices and optimizations. Here, we provide an overview of our most important contributions in [9].

We observed that $\mathbf{X_{(1)}}(\mathbf{C} \odot \mathbf{B})$ can be computed without explicitly constructing $\mathbf{C} \odot \mathbf{B}$. The theorem below solidifies our observation:

Theorem

Computing $\mathbf{X_{(1)}}(\mathbf{C} \odot \mathbf{B})$ *is equivalent to computing* $(\mathbf{N_1} * \mathbf{N_2}) \cdot \mathbf{1}_{JK}$*, where* $\mathbf{N}_1 = \mathbf{X_{(1)}} * (\mathbf{1}_I \circ (\mathbf{C}(:, f)^T \otimes \mathbf{1}_J^T))$*,* $\mathbf{N}_2 = bin(\mathbf{X_{(1)}}) * (\mathbf{1}_I \circ (\mathbf{1}_K^T \otimes \mathbf{B}(:, f)^T))$*, and* $\mathbf{1_{JK}}$ *is an all-1 vector of size* JK*, and* $f = 1 \cdots F$.

The $bin()$ function converts any nonzero value into 1. As a result, we can simplify every step of the ALS algorithm *without sacrificing accuracy*, since the above theorem states that both operations are *equivalent*. Computing $\mathbf{X_{(1)}}(\mathbf{C} \odot \mathbf{B})$ naively would require a total of $JKF + 2\text{nnz}(\mathcal{X})F$ flops, and $JKF + \text{nnz}(\mathcal{X})$ intermediate data size, where nnz($\mathcal{X}$) denotes the number of non-zeros in $\mathcal{X}$. On the other hand, the method GigaTensor requires $5\text{nnz}(\mathcal{X})F$ flops, and $max(J + \text{nnz}(\mathcal{X}), K + \text{nnz}(\mathcal{X}))$ intermediate data size. Figure 2(b) illustrates our approach.

Other contributions in [9] include:

Order of computations: By leveraging associativity properties of the algebraic operations of ALS, we chose the ordering of operations that yields the smallest number of *flops*. As an indication of how crucial this optimization is, for the knowledge base dataset that we analyze in [9], a naive ordering would incur 2.5×10^{17} *flops*, whereas the ordering that we select results in 8×10^9 *flops*.

Parallel Outer Products: Throughout the algorithm, we need to compute products of the form $\mathbf{A}^T\mathbf{A}$. We leverage row-wise matrix partitioning; we store each row of a matrix separately in HDFS, thus enabling efficient matrix self join. Using column-wise storage would render this prohibitively expensive.

Distributed Cache Multiplication: We broadcast small matrices to all reducers, thus eliminating unnecessary loading steps. This improves both the latency, as well as the size of the intermediate results produced and stored by the algorithm.

3.3 PARCUBE

On a different note, in [14], we introduce PARCUBE, an approximate, parallelizable algorithm for PARAFAC; on top of parallelizability, PARCUBE is sparsity promoting: starting from a sparse tensor, the algorithm operates, through its entire lifetime, on sparse data, and it finally produces a sparse output. In this way, we both alleviate the intermediate data explosion problem, as well as producing sparse factors, an attribute which is vital for interpretation purposes.

The algorithm, roughly, consists of the following steps:

Biased sampling: We use biased sampling to select indices from all three modes of the tensor, creating a significantly smaller tensor. The sample bias is proportional to the marginal sum for each of the three modes. We may draw multiple such samples, and indeed, we show in [14] that this improves accuracy.

Parallel decomposition on the samples: The second step includes fitting of the PARAFAC decomposition to the sample tensors, obtained from the previous step. This step can be performed entirely in parallel, offering great speedup gains.

Merging of partial results: The final step is merging the intermediate decomposition results into a full sized set of decomposition factors. In [14], we introduce the FACTORMERGE, which is pictorially represented in Fig. 3(b). The original paper contains theoretical justification of the algorithm's correctness.

Figure 3 contains a pictorial description of PARCUBE.

3.4 Results & Discoveries

Comparison against the state of the art

At the time when [9] and [14] were written, Tensor Toolbox [6] was the state of the art (and excluding the work we showcase here, still is the state of the art), hence we chose it as our baseline. Comparison of GigaTensor against the Tensor Toolbox was made, merely, to show that it was able to handle data at least two orders of magnitude larger than what the state of the art was able to.

Figure 4 illustrates the comparison of both methods with the state of the art, in terms of scalability, as well as output sparsity (for PARCUBE). Detailed comparisons can be found in the respective original papers.

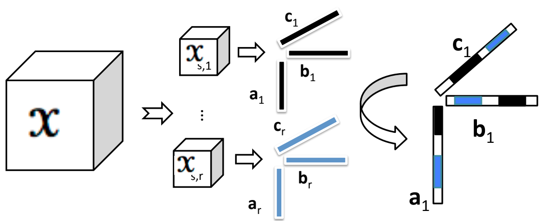
Contextual Synonym Detection.

In [9], we analyzed a knowledge base dataset, coming from the Read the Web project [1]; this dataset recorded (noun-phrase, noun-phrase, context) relationships (such as the example of Figure 1); the size of the tensor was $26M \times 26M \times 48M$, which made it prohibitive to analyze, for any existing tool. After obtaining the PARAFAC decomposition, we were able to perform contextual synonym detection, i.e. detect noun-phrases that may be used in similar contexts. Using cosine similarity, we took the low dimensional representation of each noun-phrase, as expressed by matrix $\mathbf{A}$, and we calculated the similarity of each noun-phrase to the rest. In this way, we were able to obtain noun-phrases that are contextually similar, albeit not synonyms in the traditional sense. Figure 5 contains the most notable ones.

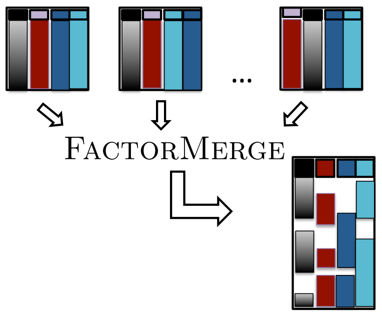
Facebook Wall posts

In [14], we analyze a Facebook wall posts dataset [1]. More specifically, the dataset we analyzed 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, resulting in a $63891 \times 63890 \times 1847$ tensor. After running PARCUBE, we stumbled

[1]Download the Facebook dataset from `http://socialnetworks.mpi-sws.org/data-wosn2009.html`

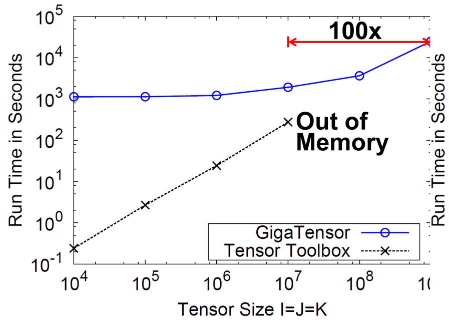


(a)

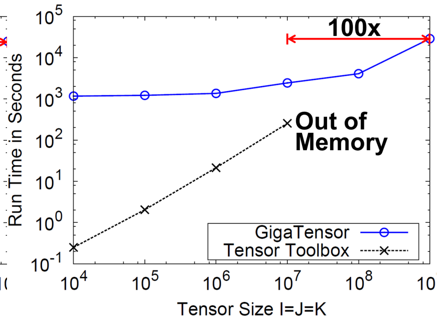


(b)

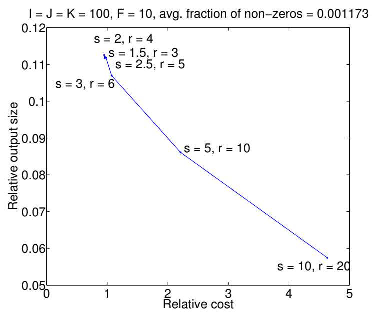
Figure 3: Subfigure (a): From the original tensor, draw r different samples, by sampling indices from each mode. It is crucial for this step, that a small subset of the drawn indices is common across different samples. Without delving into the details, this plays a major role in the third step of the algorithm, which merges partial results. For each sample tensor, compute its PARAFAC decomposition and merge the partial results. Here, for simplicity, we show a simple, rank one, case. This figure comes from our recent paper [14]. Subfigure (b): Here, we describe the merging procedure, where the rank is larger than one. From each sample tensor, we obtain a set of vectors for each mode. Let each one of the small matrices on the top represent the "sample" factor matrices. Our goal is to merge all partial components into a full-sized component, as shown at the bottom. Each color corresponds to a distinct latent component, and the upper part is marked as common across samples; notice that there are component permutations across matrices which need to be resolved in order to merge the components correctly. In [14] we provide the FACTORMERGE algorithm, as well as theoretical analysis of correctness.



(a)



(b)



(c)

Figure 4: Subfigures (a), (b): The scalability of GigaTensor compared to the Tensor Toolbox [6], for synthetic tensors of size $I \times I \times$, for two different scenarios. (a) Increasing mode dimensions, and number of nonzeros fixed and set to 10^4. (b) For a tensor of size $I \times I \times I$, increasing both the mode dimensions, and the number of nonzeros, which is set to $I/50$. In both cases, GigaTensor solves at least $100\times$ larger problem than the Tensor Toolbox which runs out of memory, for tensors of sizes beyond 10^7. We should note that in cases where the tensor fits in main memory, Tensor Toolbox is faster than GigatTensor, since it does not need to load anything from the disk (like MapReduce does). However, GigaTensor's strength is prominent when the tensor does not fit in memory, where the state of the art is unable to operate. These two subfigures are taken from our recent paper [9]. Subfigure (c): PARCUBE outputs sparse factors: Relative Output size (PARCUBE/ ALS-PARAFAC) vs Relative cost (PARCUBE PARAFAC objective function / ALS PARAFAC objective function). We see that the results of PARCUBE are more than 90% sparser than the ones from Tensor Toolbox [6], while maintaining the same approximation error. Parameters s and r are the sampling factor and the number of sampling repetitions for PARCUBE. This figure is taken from our paper [14].

(Given) Noun Phrase	(Discovered) Potential Synonyms
pollutants	dioxin, sulfur dioxide, greenhouse gases, particulates, nitrogen oxide, air pollutants, cholesterol
vodafone	verizon, comcast
Christian history	European history, American history, Islamic history, history
disbelief	dismay, disgust, astonishment

Figure 5: By decomposing a tensor of text corpus statistics, which consists of noun-phrase, context, noun-phrase triplets, we are able to identify near-synonyms of given noun-phrases. We propose to scale up this process, to form the basis for automatic discovery of new semantic categories and relations in the "Read the Web" project. This table comes from our recent paper [9].

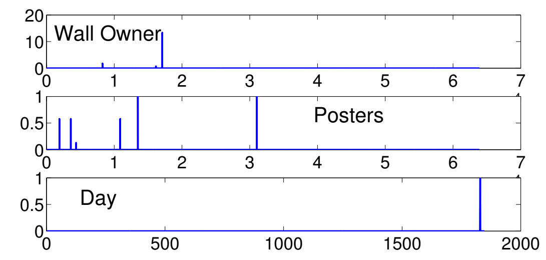


Figure 6: Facebook "anomaly": One Wall, many posters and only one day. This possibly indicates the birthday of the Wall owner. This figure is taken from our recent paper [14].

upon a series of surprising findings, an example of which we demonstrate in Figure 6: this Figure shows what appears to be the Wall owner's birthday, since many posters posted on a single day on this person's Wall; this event constitutes an "anomaly", since it deviates from normal behaviour, both intuitively, and by inspecting the majority of the decomposition components, which model the "normal" behaviour. If it hadn't been for PARCUBE's sparsity, we wouldn't be able to spot this type of anomaly without post-processing the results.

4 Insights and Conclusions

In this paper, we provided an overview of two different, successful means of scaling up tensor decompositions to big data:

GigaTensor: Scalability is achieved through simplifications of the most costly operations involved in the PARAFAC decomposition. In particular, we show how a prohibitively expensive operation can be alleviated, without sacrificing accuracy. Furthermore, we introduce a series of optimizations, which, in combination with the Map/Reduce environment, lead to a highly scalable PARAFAC decomposition algorithm.

PARCUBE: Scalability is achieved through *sketching* of the tensor, using biased sampling, parallelization of the decomposition on a number of sketches, and careful merging of the intermediate results, which, provably, produces correct PARAFAC components. Sketching might be a familiar concept in databases and data stream processing, however, in the context of large scale tensor decompositions, it has been fairly under-utilized, thus offering ample room for improvement.

The aforementioned approaches constitute, by no means, an exhaustive list of approaches one may envision in order to scale tensor decompositions up, however, we hope that they will be able to spark new research challenges and ideas, towards faster and more scalable tensor decompositions.

References

[1] Read the web. `http://rtw.ml.cmu.edu/rtw/`.

[2] E. Acar, C. Aykut-Bingol, H. Bingol, R. Bro, and B. Yener. Multiway analysis of epilepsy tensors. *Bioinformatics*, 23(13):i10–i18, 2007.

[3] C.A. Andersson and R. Bro. The n-way toolbox for matlab. *Chemometrics and Intelligent Laboratory Systems*, 52(1):1–4, 2000.

[4] 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.

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

[6] B.W. Bader and T.G. Kolda. Matlab tensor toolbox version 2.2. *Albuquerque, NM, USA: Sandia National Laboratories*, 2007.

[7] R. Bro. Parafac. tutorial and applications. *Chemometrics and intelligent laboratory systems*, 38(2):149–171, 1997.

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

[9] U. Kang, E. Papalexakis, A. Harpale, and C. Faloutsos. Gigatensor: scaling tensor analysis up by 100 times-algorithms and discoveries. In *SIGKDD*, pages 316–324. ACM, 2012.

[10] H.A.L. Kiers. Towards a standardized notation and terminology in multiway analysis. *Journal of Chemometrics*, 14(3):105–122, 2000.

[11] 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.

[12] T.G. Kolda and B.W. Bader. Tensor decompositions and applications. *SIAM review*, 51(3), 2009.

[13] 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.

[14] E. Papalexakis, C. Faloutsos, and N. Sidiropoulos. Parcube: Sparse parallelizable tensor decompositions. *Machine Learning and Knowledge Discovery in Databases*, pages 521–536, 2012.

[15] 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.

[16] J. Sun, S. Papadimitriou, C.Y. Lin, N. Cao, S. Liu, and W. Qian. Multivis: Content-based social network exploration through multi-way visual analysis. In *Proc. SDM*, volume 9, pages 1063–1074, 2009.

[17] 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.

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

[19] Q. Zhang, M. Berry, B. Lamb, and T. Samuel. A parallel nonnegative tensor factorization algorithm for mining global climate data. *Computational Science–ICCS 2009*, pages 405–415, 2009.