PARCUBE: Sparse Parallelizable CANDECOMP-PARAFAC Tensor Decomposition

Evangelos E. Papalexakis, Carnegie Mellon University
Christos Faloutsos, Carnegie Mellon University
Nicholas D. Sidiropoulos, University of Minnesota

How can we efficiently decompose a tensor into sparse factors, when the data does not fit in memory? Tensor decompositions have gained a steadily increasing popularity in data mining applications, however the current state-of-art decomposition algorithms operate on main memory and do not scale to truly large datasets. In this work, we propose PARCUBE, a new and highly parallelizable 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. We provide theoretical guarantees for the algorithm's correctness and we experimentally validate our claims through extensive experiments, including four different real world datasets (ENRON, LBNL, FACEBOOK and NELL), demonstrating its effectiveness for data mining practitioners. In particular, we are the first to analyze the very large NELL dataset using a sparse tensor decomposition, demonstrating that PARCUBE enables us to handle effectively and efficiently very large datasets. Finally, we make our highly scalable parallel implementation publicly available, enabling reproducibility of our work.

Additional Key Words and Phrases: Tensors, PARAFAC decomposition, sparsity, sampling, randomized algorithms, parallel algorithms

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 [Bader and Kolda 2007a; Andersson and Bro 2000] 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 [Drineas et al. 2006], 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 [Drineas et al. 2006]; this method has also been extended in order to handle tensors [Mahoney et al. 2006]. However, both these methods are tied to a specific decomposition, while we desire to disconnect sampling from the specific decomposition that follows.

This paper 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

Author's addresses:

DOI : http://dx.doi.org/10.1145/0000000.0000000

applications. For instance, PARCUBE produces over 90% sparser results than regular PARAFAC, while maintaining the same approximation error.

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 [Papalexakis et al. 2012]. In this extended version, in addition to the contributions of [Papalexakis et al. 2012], 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 finally, we provide a very efficient parallel implementation which we make publicly available at http://www.cs.cmu.edu/~epapalex/src/parCube.zip, enabling *reproducibility* of PARCUBE.

The rest of this paper is structured as follows. Section 2 provides some useful background; section 3 describes the proposed method, and section 4 contains experiments. We review related work in section 5, and conclusions are drawn in section 6.

2. TENSOR DECOMPOSITIONS

Notation preliminaries

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. Table I shows the symbols used, in compact form.

Tensors A tensor of n modes (or n-way/n-mode tensor) is a structure indexed by n variables. For example, a matrix is a two-way tensor. In this work, we focus on three-way tensors, because they are most common; however, all results can be readily extended to higher-way tensors. A three-way tensor $\underline{\mathbf{X}}$ is a structure that resembles a data cube. A detailed survey for tensors and tensor decompositions may be found in [Kolda and Bader 2009].

Table I. Table of symbols

Symbol	Definition
$\underline{\mathbf{X}}$	three-way tensor
$\mathbf{A}$	matrix
$\mathbf{x}$	column vector
a	scalar
I	index
NNZ( )	number of non-zeros

2.1. The PARAFAC decomposition

The PARAFAC decomposition [Harshman 1970] of $\underline{\mathbf{X}}$ into F components is

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

where $\mathbf{a} \circ \mathbf{b} \circ \mathbf{c}(i, j, k) = \mathbf{a}(i)\mathbf{b}(j)\mathbf{c}(k)$. A pictorial exaple of PARAFAC is shown on Fig. 1

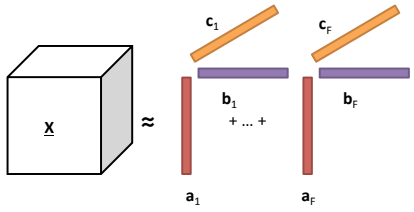


Fig. 1. The F-component PARAFAC decomposition of $\underline{\mathbf{X}}$.

Essentially, in order to obtain the PARAFAC decomposition, we need to solve the following optimization problem:

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C}} \|\underline{\mathbf{X}} - \sum_{f=1}^{F} \mathbf{a}_f \circ \mathbf{b}_f \circ \mathbf{c}_f\|_F^2 \tag{1}$$

The most popular algorithm for fitting the PARAFAC decomposition is the Alternating Least Squares (ALS) [Bro 1997; Kolda and Bader 2009]. The computational complexity of the ALS Algorithm for a $I \times J \times K$ tensor, and for F components is $O(IJKF)$ per iteration.

2.2. Tensor compression & TUCKER3

Consider the $I \times J \times K$ tensor $\underline{\mathbf{X}}$. Then, its $\{Q, R, P\}$ TUCKER3 decomposition consists of a $P \times Q \times R$ core tensor, say, $\underline{\mathbf{G}}$ and three assorted, unitary, matrices $\mathbf{U}, \mathbf{V}, \mathbf{W}$ with sizes $I \times P$, $J \times Q$ and $K \times R$ respectively. The decomposition can be compactly written as

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

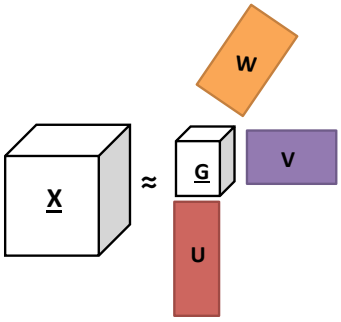


Fig. 2. The TUCKER3 decomposition of $\underline{\mathbf{X}}$.

A pictorial representation of TUCKER3 is shown in Fig. 2. TUCKER3 is very useful for compression purposes: if we choose $P \ll I$, $Q \ll J$, and $R \ll K$, then, we get a core tensor $\underline{\mathbf{G}}$ which is a compressed version of $\underline{\mathbf{X}}$. This approach is used in practice [Bro et al. 1999] in order to speed up further operations in a tensor, since one is able to roll-back from the compressed tensor to the original $\underline{\mathbf{X}}$ using the factor matrices $\mathbf{U}, \mathbf{V}, \mathbf{W}$.

3. THE PARCUBE METHOD

In this section we introduce PARCUBE, a new method for PARAFAC decomposition designed with 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.

3.1. 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 [1]. 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

[1] Another, reasonable alternative is the sum-of-squares of the elements of rows, columns and fibers, which is a measure of *energy*. We leave this for future work.

$p_{\mathcal{J}}(j) = \mathbf{x}_b(j)/\sum_{j=1}^{J} \mathbf{x}_b(j)$. 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 1 provides an outline of the sampling forPARCUBE.

LEMMA 3.1. *The computational complexity of Algorithm 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 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 $\underline{\mathbf{X}}_s = \underline{\mathbf{X}}(\mathcal{I}, \mathcal{J}, \mathcal{K})$.

3.2. 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 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 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 3 which is our main contribution. Algorithm 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 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. In section 3.3, 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 3.2 provides a first correctness result for our merging algorithm.

3.3. Merging explained

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.

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 3.4 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). 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 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 2. In the next iteration of Algorithm 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 3.4. 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.

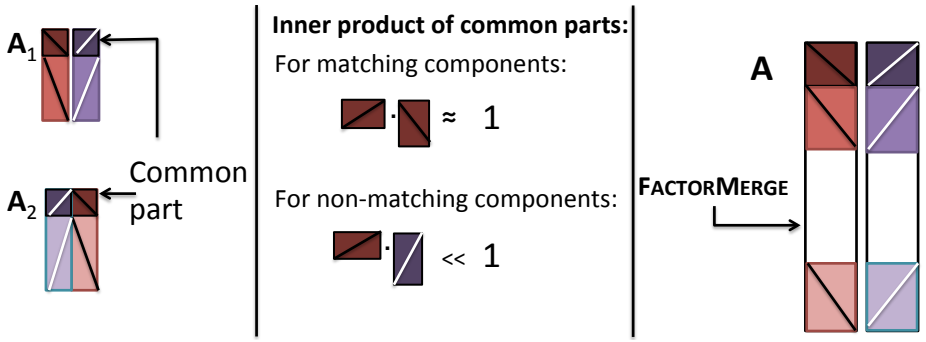


Fig. 3. Example of merging two partial factor matrices to the final matrix, while accounting for potential column permutations. (Best viewed in color)

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.

An illustrative example of our merging scheme, for two partial factor matrices, is shown in Fig. 3. 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. 3. Then, Algorithm 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 by $\mathbf{A}_1$ (shown in different colors in Fig. 3) is the order that Algorithm 4 will impose to the rest of the partial results. In Fig. 3, 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 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 very similar to the very well know Hungarian Algorithm [Kuhn 1955], used to efficiently solve combinatorial assignment problems.

Algorithm 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 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: $\mathbf{A}(\mathcal{I}, :) = \mathbf{A}_s$, $\mathbf{B}(\mathcal{J}, :) = \mathbf{B}_s$, $\mathbf{C}(\mathcal{K}, :) = \mathbf{C}_s$

Algorithm 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 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 ℓ_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: 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 $I \times F$.

Set $\mathbf{A} = \mathbf{A}_1$
for $i = 2 \cdots r$ **do**
  for $f_1 = 1 \cdots F$ **do**
    for $f_2 = 1 \cdots F$ **do**
      Compute similarity $\mathbf{v}(f_2) = (\mathbf{A}(\mathcal{I}_p, f_2))^T (\mathbf{A}_i(\mathcal{I}_p, f_1)))$
    end for
    $c = \arg\max_{c'} \mathbf{v}(c')$
    Update only the zero entries of $\mathbf{A}(:, c)$ using vector $\mathbf{A}_i(:, f_1)$.
  end for
end for

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

PROPOSITION 3.2. *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 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*

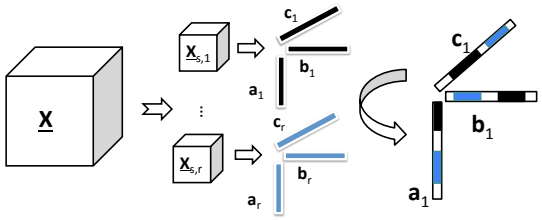


Fig. 4. Example of rank-1 PARAFAC using PARCUBE (Algorithm 3). The procedure described is the following: Create r independent samples of $\underline{\mathbf{X}}$, using Algorithm 1. Run the PARAFAC- ALS algorithm for $K = 1$ and obtain r triplets of vectors, corresponding to the first component of $\underline{\mathbf{X}}$. As a final step, combine those r triplets, by distributing their values to the original sized triplets, as indicated in Algorithm 3.

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 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 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 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 to correctly merge the pieces coming from the different samples. We have carried out extensive experiments verifying that Algorithm 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 [Sidiropoulos et al. 2014a; Sidiropoulos et al. 2014b] we propose 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.

3.5. 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 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.

3.6. 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., [Papalexakis et al. 2013]), 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 4.2, PARCUBE's solution will converge to the solution that directly optimizes for sparsity.

3.7. Extension to other models

Even though the focus of the present paper is the PARAFAC decomposition, the same methodology can be applied in order to accelerate and parallelize other tensor decomposition models. For instance, in [Papalexakis et al. 2014], 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 TUCKER3 is not straightforward. In Section 3.4 we invoke the uniqueness of the PARAFAC factors in order to show that the merging will be correct; however, TUCKER3 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 TUCKER3, but simply that this needs to be done carefully, in light of the differences of TUCKER3 from PARAFAC.

4. EXPERIMENTS & DISCOVERIES

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 [Bader and Kolda 2007a]. 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 [Papalexakis and Sidiropoulos 2011], which is the state of the art for PARAFAC decompositions with sparsity on the latent factors. The results of Section 4.1 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 TUCKER3 compression accelerated PARAFAC decomposition. The aforementioned experiments correspond to Sections 4.3 - 4.6 and were carried out on a machine with 4 Intel Xeon E74850 2.00GHz, and 512Gb of RAM.

Finally, in Section 4.7 we apply our approach in order to analyze real datasets presented in Table II.

Table II. Datasets analyzed

Name	Description	Dimensions	NNZ	Sparsity ($\frac{NNZ}{IJK}$)
ENRON [ENR 2014]	(sender, recipient, month)	$186 \times 186 \times 44$	9838	0.0065
LBNL [Pang et al. 2005]	(src, dst, port #)	$65170 \times 65170 \times 65327$	27269	10^{-10}
FACEBOOK [Viswanath et al. 2009]	(wall owner, poster, day)	$63891 \times 63890 \times 1847$	737778	10^{-7}
NELL [RTW 2014]	(noun-phrase, noun-phrase, context)	$14545 \times 14545 \times 28818$	76879419	10^{-5}

We implemented PARCUBE in Matlab and Java, and we make it *publicly available*[2]. We furthermore use the Tensor Toolbox for Matlab [Bader and Kolda 2007a] as our core PARAFAC decomposition implementation.

4.1. Performance & Speedup Evaluation

In the following lines, we evaluate the performance of PARAFAC using PARCUBE (Algorithm 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 1 for the approximation cost of PARAFAC. In Fig. 5, 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. 6, 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. 6, the approximation error improves as the number of repetitions r increases, as expected from our theoretical analysis of Section 3.4.

4.2. Factor Sparsity Assessment

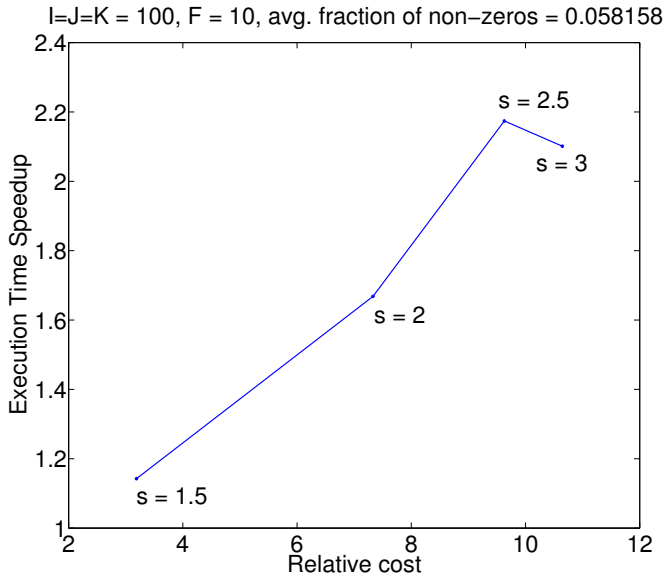
In Fig.7, we measure the relative output size (i.e. the relative degree of sparsity) between PARCUBE and Tensor Toolbox non-negative PARAFAC. As before, we carried out 50 iterations of the experiment and report mean values. The output size is simply defined as $NNZ(\mathbf{A}) + NNZ(\mathbf{B}) + NNZ(\mathbf{C})$, 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. 8 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 [Papalexakis and Sidiropoulos 2011; Papalexakis et al. 2013] [3]. This further provides evidence on the validity of the sparsity introduced by PARCUBE.

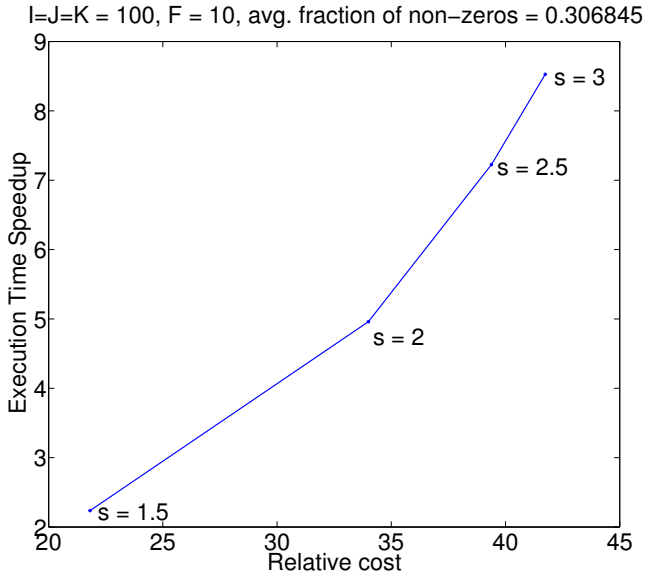
4.3. Parallelizability

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.

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

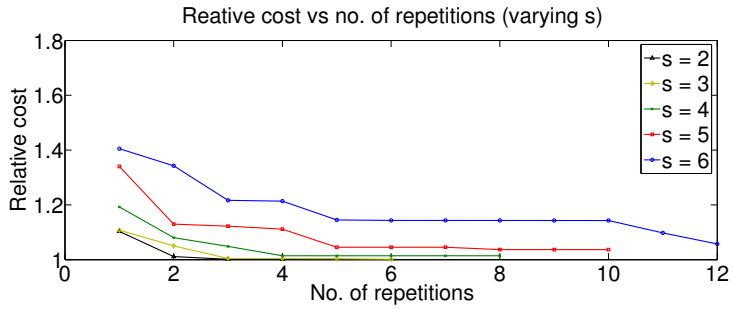


(a)

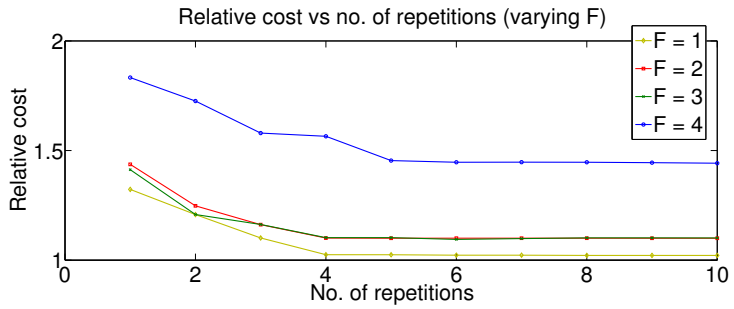


(b)

Fig. 5. 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 cost vs No. of repetitons (varying s)



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

Fig. 6. PARCUBE reduces the PARAFAC approximation cost: (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

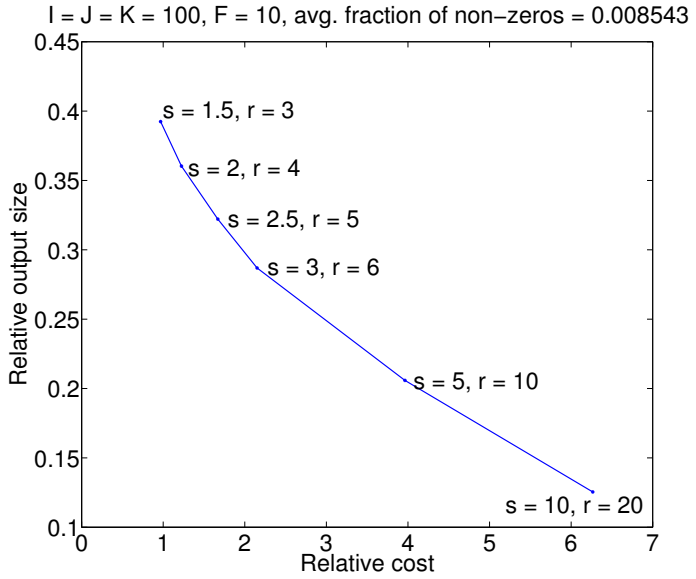
Figure 9 shows our results: Subfigure 9(a) contains the speedup for different number of parallel workers, as the number of non-zeros increases. We observe that when as the number of non-zeros increases, the speedup due to parallelizability becomes more pronounced; intuitively, this result makes sense, 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 9(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.

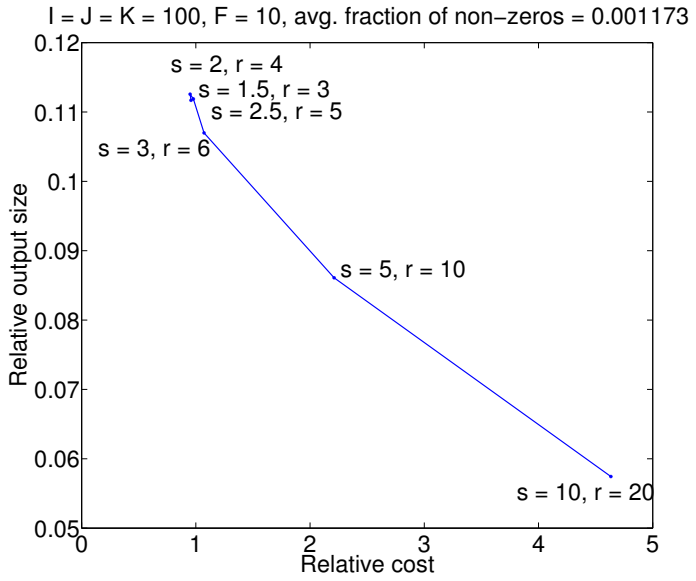
4.4. Scalability in terms of data & parameter size

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. 10, 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.



(a)



(b)

Fig. 7. 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.

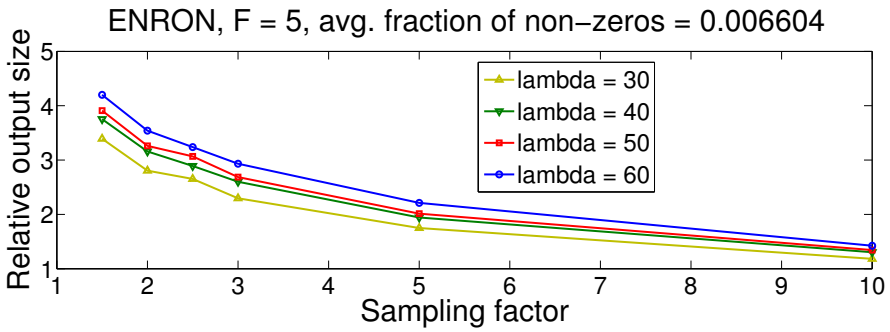


Fig. 8. PARCUBE outputs sparse factors: Relative Output size (PARCUBE/ PARAFAC SLF) vs sampling factor s (where no. of repetitions is $r = 2s$.

(2) **Input dimensionality** (measured in the mode sizes of the tensor): In Fig. 11, 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. 12 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. 12) 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.

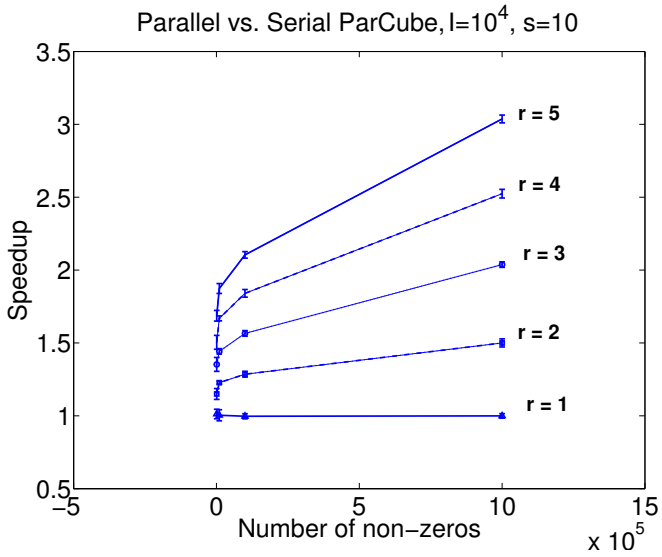
We ran the above experiments 5 independent times, and as the error-bars indicate, the variability of the results is minimal, thus PARCUBE is consistent in terms of scalability.

4.5. Accuracy as a function of tensor density

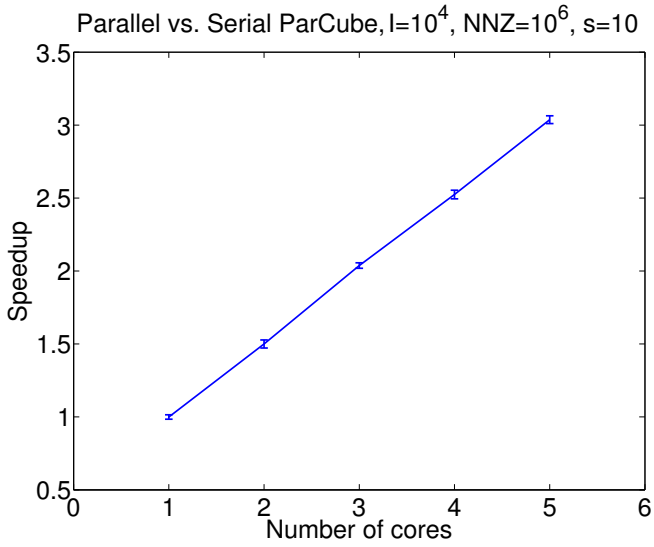
Here, we experimentally demonstrate that PARCUBE's performance is consistent for tensors of varying density. In particular, we created a series of randomly generated $10^2 \times 10^2 \times 10^2$ tensors, with varying number of non-zeros, ranging from fully dense (i.e. 10^6 non-zeros), to 0.99 percent sparse (10^4 non-zeros). In order to estimate the stability of our results, we ran 50 independent iterations of this experiment, for all different tensors. In Figure 13 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.

4.6. Comparison against TUCKER3 compression

As we highlighted in Section 2.2, an alternative approach of reducing the size of the tensor into a smaller, compressed version is via the TUCKER3 decomposition. We compare against the method introduced in [Bro et al. 1999], where the tensor is first compressed using TUCKER3, PARAFAC is fitted in the compressed data and the factor matrices of the TUCKER3 model are used to decompress the results. In order to compute the TUCKER3 decomposition, we use the highly optimized Memory Efficient Tucker (MET) algorithm [Kolda and Sun 2008], included in the Tensor Toolbox for Matlab



(a)



(b)

Fig. 9. Serial vs. Parallel PARCUBE

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 $P = Q = R = 100$ for TUCKER3, while the original tensor is of dimensions $10^4 \times 10^4 \times 10^4$.

Figure 14 clearly shows that PARCUBE is orders of magnitude faster than TUCKER3 compression. The reason why this behavior is observed is because computing the TUCKER3 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

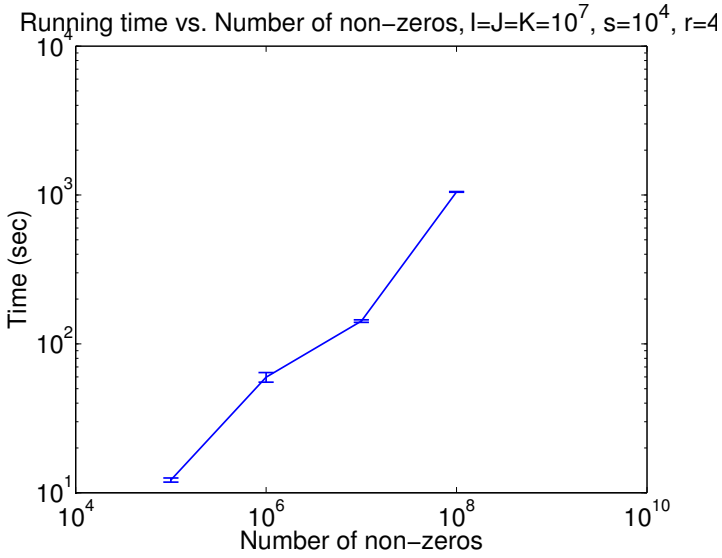


Fig. 10. Scalability with respect to the number of non-zeros.

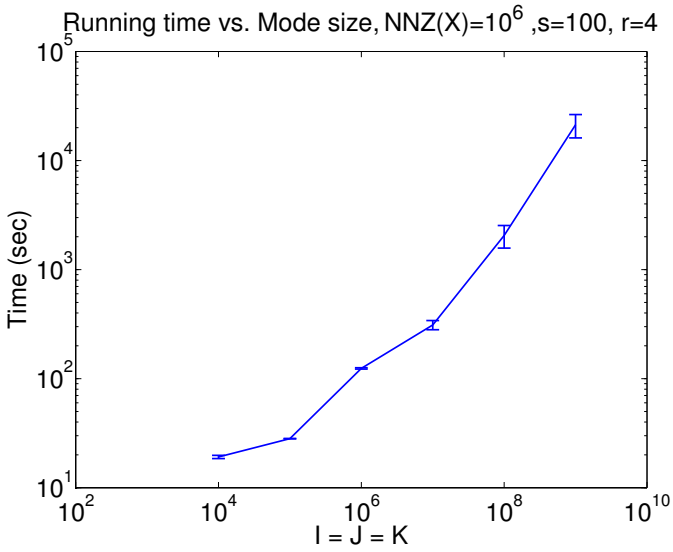


Fig. 11. Scalability with respect to the tensor dimensions $I = J = K$.

highly optimized algorithm such as MET. On the other hand, the sampling step of PARCUBE is, in practice, much faster than computing the TUCKER3 decomposition on the full data, and thus PARCUBE ends up being significantly faster.

4.7. PARCUBE at work

In this section we present interesting patterns and anomalies, that we were able to discover in the datasets of Table II, demonstrating that our proposed algorithm PARCUBE is both practical and

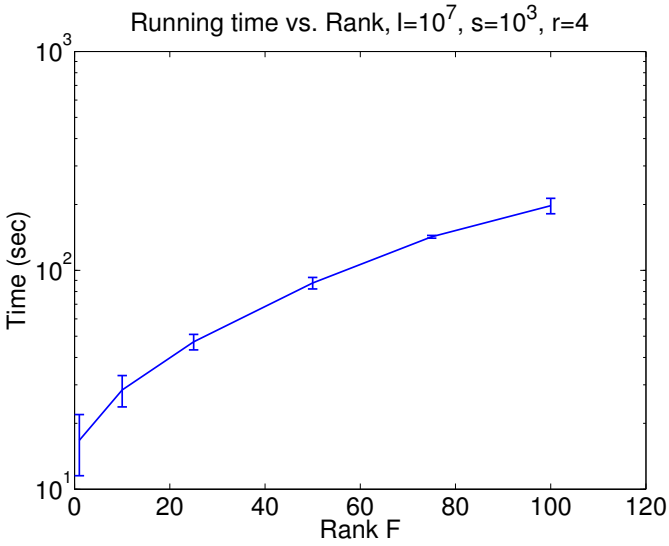


Fig. 12. Scalability with respect to the decomposition rank.

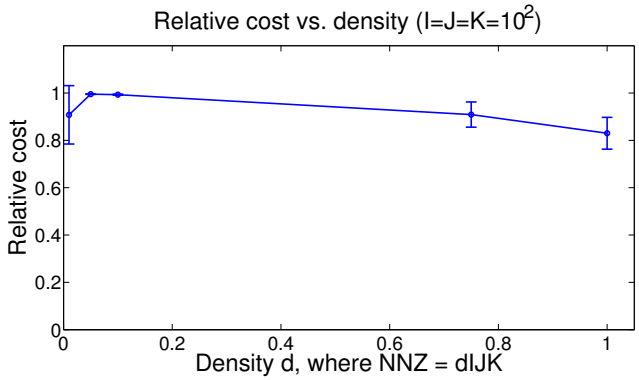


Fig. 13. Relative errors as a function of the density of the tensor, for $F = 3$, sampling factor $s = 2$ and $r = 10$ repetitions. The density d is defined as $NNZ(\underline{\mathbf{X}}) = dIJK$.

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.

4.7.1. **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 3 using $s = 2$ and $r = 4$ and we applied similar analysis to the resulting factors as the one applied in [Bader et al. 2006; Papalexakis and Sidiropoulos 2011]. In Figure 15 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

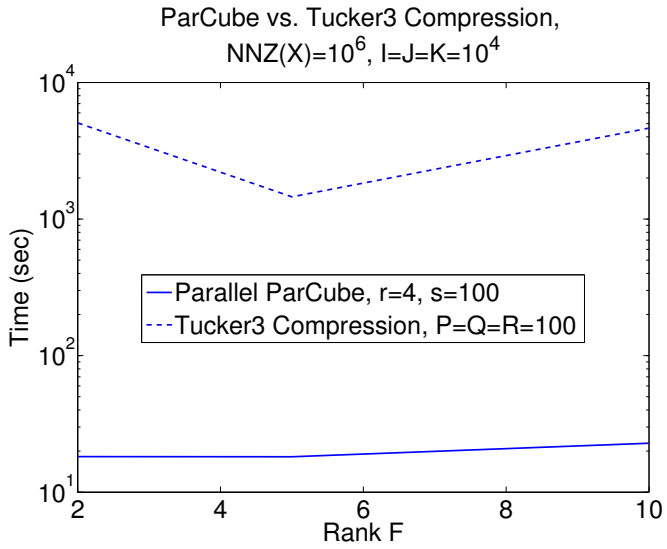


Fig. 14. Parallel PARCUBE against TUCKER3 compression accelerated PARAFAC. PARCUBE is orders of magnitude faster.

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.

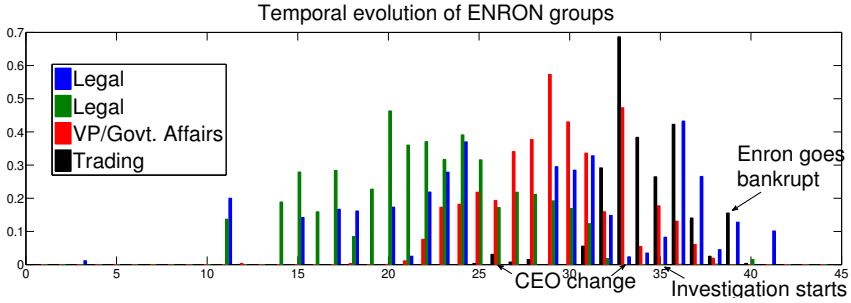


Fig. 15. Temporal evolution of 4 groups in the ENRON dataset. We have labelled the groups, according to the position of the participants in the company. The labels of the extracted groups are consistent with other works in the literature albeit they have been extracted with somewhat different order. 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.

4.7.2. **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 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. 16 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.

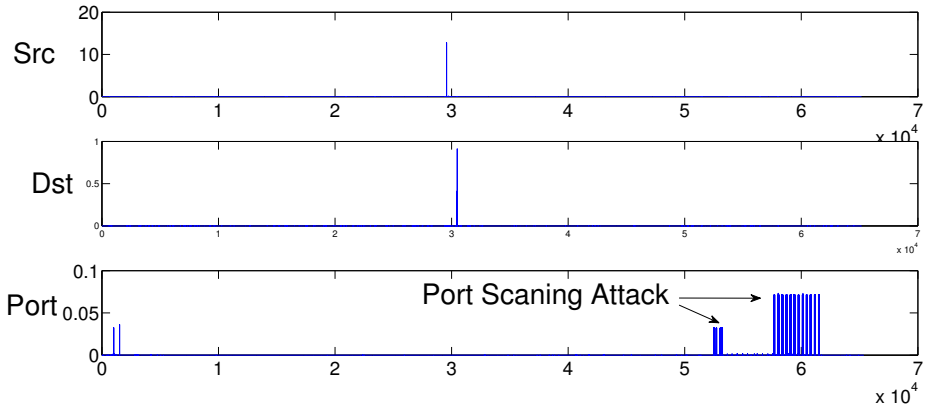


Fig. 16. Anomaly on the LBNL data: We have one source address (addr. 29571), contacting one destination address (addr. 30483) using a wide range of near-consecutive ports, possibly indicating a port scanning attack.

4.7.3. **FACEBOOK Wall posts**. This dataset [4] first appeared in [Viswanath et al. 2009]; 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 3 using $s = 100$ and $r = 10$. In Figure 17 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.

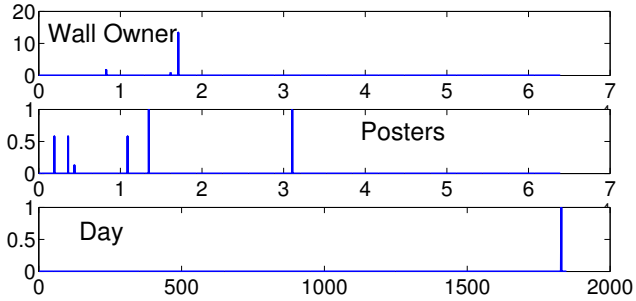
4.7.4. **NELL**. This dataset consists of triplets of the form (noun-phrase, noun-phrase, context). which form a tensor with assorted modes of size $14545 \times 14545 \times 28818$ and 76879419 non-zeros, and as values the number of occurrences of each triplet. The context phrase may be just a verb 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:

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

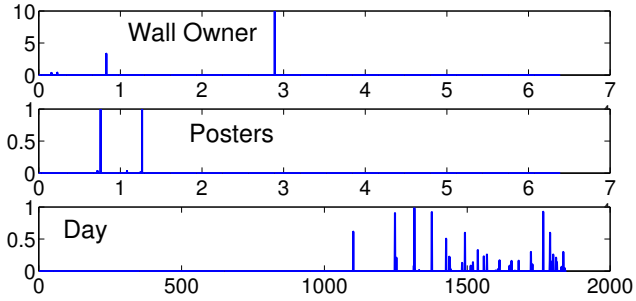
We were not able to compute the exact PARAFAC decomposition on a single machine, and thus, we estimate the number of non-zeros of a fully dense matrix $\mathbf{A}$ as IF (and accordingly for the remaining factors).

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 III; the relationship between the words in that table can be viewed as being contextually similar. Additionally, in Table IV, 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,

[4] Download FACEBOOK at http://socialnetworks.mpi-sws.org/data-wosn2009.html



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



(b) FACEBOOK normal activity

Fig. 17. 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

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

Table III. NELL: Potential synonym discovery

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 IV. NELL: Concepts of noun-phrases and context words

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*

5. RELATED WORK

5.1. Tensor applications

Tensors and tensor decompositions have gained increasing popularity in the last few years, in the data mining community [Kolda and Bader 2009]. The list of tensor applications in data mining is long, however we single out a few that we deemed representative: In [Kolda and Bader 2006], the authors extend the well known link analysis algorithm HITS, incorporating textual/topical information. In [Bader et al. 2006] and [Bader et al. 2008] the authors use tensors for social network analysis on the ENRON dataset. In [Sun et al. 2009], the authors propose a sampling-based TUCKER3 decomposition in order to perform content based network analysis and visualization. The list continues, including applications such as Cross-language Information Retrieval [Chew et al. 2007], Anomaly Detection [Maruhashi et al. 2011], Brain Signal Analysis and detection of Epilepsy[Acar et al. 2007], Machine Vision [Vasilescu and Terzopoulos 2002], Web Search[Sun et al. 2005], and Bioinformatics [Li and Ngom 2011], to name a few. Apart from Data Mining, tensors have been and are still being applied in a multitude of fields such as Chemometrics [Bro 1997] and Signal Processing [Sidiropoulos et al. 2000].

5.2. State of the art toolboxes

The standard framework for working with tensors is Matlab; there exist two toolboxes, both of very high quality: The Tensor Toolbox for Matlab[Bader and Kolda 2007b; Bader and Kolda 2007a] (specializing in sparse tensors) and the N-Way Toolbox for Matlab [Andersson and Bro 2000] (specializing in dense tensors).

5.3. Fast and scalable tensor decompositions

The authors of [Bro et al. 1999] introduce an algorithm that compresses the tensor using TUCKER3, and decomposes the core tensor, which is significantly smaller. In [Phan and Cichocki 2009; Huy Phan and Cichocki 2011], the authors propose a partition-and-merge scheme for the PARAFAC decomposition which, however, does not offer factor sparsity. In [Papalexakis and Sidiropoulos 2011], the authors introduce a PARAFAC decomposition with latent factor sparsity. In [Nion and Sidiropoulos 2009] and [Sun et al. 2006] we find two interesting approaches, where a tensor is viewed as a stream and the challenge is to track the decomposition. In terms of parallel algorithms, [Zhang et al. 2009] introduces a parallel Non-negative Tensor Factorization. In [Tsourakakis 2009; Sun et al. 2009] the authors propose randomized, sampling based TUCKER3 decompositions. In [Kang et al. 2012], the authors introduce a highly scalable Alternating Least Squares implementation of PARAFAC for Hadoop, whereas in [Beutel et al. 2014], the authors provide a versatile and highly scalable Distributed Stochastic Gradient Descent Hadoop implementation which, among others, is able to perform PARAFAC decomposition. In [Kim and Candan 2011] and [Kim and Candan 2012] the authors introduce a very interesting, alternative viewpoint of scaling up tensor decompositions, employing relational algebra. The authors of [De Almeida et al. 2014] provide an alternative framework of parallelizing tensor decompositions, based on partitioning the problem and distributing the computation using a multi-layer graph in order to represent the machines that operate on the

problem. In [Papalexakis et al. 2014], the authors build upon the present method in order to speed up the related problem of Coupled Matrix-Tensor Factorization, achieving a speedup factor of 200. In [Sidiropoulos et al. 2014a; Sidiropoulos et al. 2014b], following a similar approach of reducing the dimensions of the tensor and decomposing smaller instances of the problem in parallel, the authors introduce an algorithm that uses random projections and they provide identifiability guarantees.

6. CONCLUSION

In this work 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 highly scalable parallel implementation (which is publicly available) that scales for very large tensors. Finally, we highlight the practicality of PARCUBE by analyzing four different real datasets, discovering patterns and anomalies.

7. ACKNOWLEDGEMENTS

Research was sponsored by the National Science Foundation, Grant No, IIS-1247489 and IIS-1247632, and the Defense Threat Reduction Agency under contract No. HDTRA1-10-1-0120. 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. We would also like to thank the anonymous reviewers for their critical and creative comments and suggestions.

REFERENCES

Last accessed: 9/9/2014. ENRON E-mail dataset. http://www.cs.cmu.edu/~enron/. (Last accessed: 9/9/2014).

Last accessed: 9/9/2014. Read the Web. http://rtw.ml.cmu.edu/rtw/. (Last accessed: 9/9/2014).

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

C.A. Andersson and R. Bro. 2000. The N-way toolbox for MATLAB. *Chemometrics and Intelligent Laboratory Systems* 52, 1 (2000), 1–4.

B.W. Bader, M.W. Berry, and M. Browne. 2008. Discussion tracking in Enron email using PARAFAC. *Survey of Text Mining II* (2008), 147–163.

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

B.W. Bader and T.G. Kolda. 2007a. Matlab tensor toolbox version 2.2. *Albuquerque, NM, USA: Sandia National Laboratories* (2007).

Brett W. Bader and Tamara G. Kolda. 2007b. Efficient MATLAB computations with sparse and factored tensors. *SIAM Journal on Scientific Computing* 30, 1 (December 2007), 205–231.

Alex Beutel, Abhimanu Kumar, Evangelos Papalexakis, Partha Pratim Talukdar, Christos Faloutsos, and Eric P Xing. 2014. FLEXIFACT: Scalable Flexible Factorization of Coupled Tensors on Hadoop. (2014).

R. Bro. 1997. PARAFAC. Tutorial and applications. *Chemometrics and intelligent laboratory systems* 38, 2 (1997), 149–171.

R Bro, ND Sidiropoulos, and GB Giannakis. 1999. A fast least squares algorithm for separating trilinear mixtures. In *Int. Workshop Independent Component and Blind Signal Separation Anal*. 11–15.

P.A. Chew, B.W. Bader, T.G. Kolda, and A. Abdelali. 2007. Cross-language information retrieval using PARAFAC2. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 143–152.

André LF De Almeida, Alain Y Kibangou, and others. 2014. Distributed Large-Scale Tensor Decomposition. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*.

P. Drineas, R. Kannan, M.W. Mahoney, and others. 2006. Fast Monte Carlo algorithms for matrices III: Computing a compressed approximate matrix decomposition. *SIAM J. Comput.* 36, 1 (2006), 184.

R.A. Harshman. 1970. Foundations of the PARAFAC procedure: Models and conditions for an" explanatory" multimodal factor analysis. (1970).

Anh Huy Phan and Andrzej Cichocki. 2011. PARAFAC algorithms for large-scale problems. *Neurocomputing* 74, 11 (2011), 1970–1984.

U Kang, Evangelos Papalexakis, Abhay Harpale, and Christos Faloutsos. 2012. 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*. ACM, 316–324.

Mijung Kim and Kasim Selçuk Candan. 2011. Approximate tensor decomposition within a tensor-relational algebraic framework. In *Proceedings of the 20th ACM international conference on Information and knowledge management*. ACM, 1737–1742.

Mijung Kim and K Selçuk Candan. 2012. 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*. ACM, 355–364.

T.G. Kolda and B.W. Bader. 2006. The TOPHITS model for higher-order web link analysis. In *Workshop on Link Analysis, Counterterrorism and Security*, Vol. 7. 26–29.

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

Tamara G Kolda and Jimeng Sun. 2008. Scalable tensor decompositions for multi-aspect data mining. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*. IEEE, 363–372.

Harold W Kuhn. 1955. The Hungarian method for the assignment problem. *Naval research logistics quarterly* 2, 1-2 (1955), 83–97.

Yifeng Li and Alioune Ngom. 2011. Classification of clinical gene-sample-time microarray expression data via tensor decomposition methods. In *Computational Intelligence Methods for Bioinformatics and Biostatistics*. Springer, 275–286.

M.W. Mahoney, M. Maggioni, and P. Drineas. 2006. Tensor-CUR decompositions for tensor-based data. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 327–336.

K. Maruhashi, F. Guo, and C. Faloutsos. 2011. 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*.

D. Nion and N.D. Sidiropoulos. 2009. Adaptive algorithms to track the PARAFAC decomposition of a third-order tensor. *Signal Processing, IEEE Transactions on* 57, 6 (2009), 2299–2310.

R. Pang, M. Allman, M. Bennett, J. Lee, V. Paxson, and B. Tierney. 2005. A first look at modern enterprise traffic. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*. USENIX Association, 2–2.

E.E. Papalexakis and N.D. Sidiropoulos. 2011. Co-clustering as multilinear decomposition with sparse latent factors. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*. IEEE, 2064–2067.

Evangelos E Papalexakis, Christos Faloutsos, and Nicholas D Sidiropoulos. 2012. Parcube: Sparse parallelizable tensor decompositions. In *Machine Learning and Knowledge Discovery in Databases*. Springer, 521–536.

Evangelos E Papalexakis, Tom M Mitchell, Nicholas D Sidiropoulos, Christos Faloutsos, Partha Pratim Talukdar, and Brian Murphy. 2014. Turbo-SMT: Accelerating Coupled Sparse Matrix-Tensor Factorizations by 200x. In *SIAM SDM*.

Evangelos E Papalexakis, Nicholas D Sidiropoulos, and Rasmus Bro. 2013. From K-means to higher-way co-clustering: multilinear decomposition with sparse latent factors. *Signal Processing, IEEE Transactions on* 61, 2 (2013), 493–506.

A.H. Phan and A. Cichocki. 2009. Block decomposition for very large-scale nonnegative tensor factorization. In *Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP), 2009 3rd IEEE International Workshop on*. IEEE, 316–319.

N.D. Sidiropoulos, G.B. Giannakis, and R. Bro. 2000. Blind PARAFAC receivers for DS-CDMA systems. *Signal Processing, IEEE Transactions on* 48, 3 (2000), 810–823.

ND Sidiropoulos, EE Papalexakis, and C Faloutsos. 2014a. A Parallel Algorithm for Big Tensor Decomposition Using Randomly Compressed Cubes (PARACOMP). In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*.

N Sidiropoulos, E Papalexakis, and C Faloutsos. 2014b. Parallel Randomly Compressed Cubes: A scalable distributed architecture for big tensor decomposition. *Signal Processing Magazine, IEEE* 31, 5 (2014), 57–70.

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

J. Sun, D. Tao, and C. Faloutsos. 2006. Beyond streams and graphs: dynamic tensor analysis. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 374–383.

J.T. Sun, H.J. Zeng, H. Liu, Y. Lu, and Z. Chen. 2005. CubeSVD: a novel approach to personalized Web search. In *Proceedings of the 14th international conference on World Wide Web*. ACM, 382–390.

C.E. Tsourakakis. 2009. Mach: Fast randomized tensor decompositions. *Arxiv preprint arXiv:0909.4969* (2009).

M. Vasilescu and D. Terzopoulos. 2002. Multilinear analysis of image ensembles: Tensorfaces. *Computer Vision ECCV 2002* (2002), 447–460.

Bimal Viswanath, Alan Mislove, Meeyoung Cha, and Krishna P. Gummadi. 2009. On the Evolution of User Interaction in Facebook. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Social Networks (WOSN'09)*.

Q. Zhang, M. Berry, B. Lamb, and T. Samuel. 2009. A Parallel Nonnegative Tensor Factorization Algorithm for Mining Global Climate Data. *Computational Science–ICCS 2009* (2009), 405–415.