ParCube: Sparse Parallelizable Tensor Decompositions

Evangelos E. Papalexakis[1] ⋆, Christos Faloutsos[1], and Nicholas D. Sidiropoulos ⋆⋆[2]

[1] School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA, (epapalex,christos)@cs.cmu.edu

[2] Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN, USA, nikos@ece.umn.edu

Abstract. 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.

Keywords: 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

⋆ Research was sponsored by the Defense Threat Reduction Agency and was accomplished under contract No. HDTRA1-10-1-0120. Funding was provided by the U.S. Army Research Office (ARO) and Defense Advanced Research Projects Agency (DARPA) under Contract Number W911NF-11-C-0088. Research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-09-2-0053. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on. 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.

⋆⋆ N. Sidiropoulos was partially supported by ARO contract W911NF-11-1-0500.

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 [8,4] 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 [11], 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 [11]; this method has also been extended in order to handle tensors [15]. 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 applications. 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.

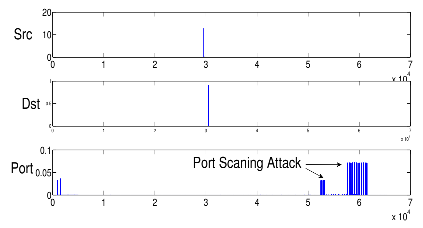
In Fig. 1 we demonstrate a preview of our results: On subfigure 1(a), we show a successful discovery of what appears to be a *port scanning attack*, on the LBNL network traffic dataset, and subfigure 1(b) demonstrates over 90% sparser results than regular Parafac, while maintaining the same approximation error.

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. Related work is reviewed in section 5, and conclusions are drawn in section 6.

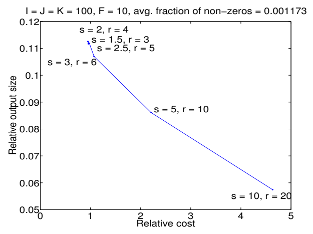
2 Tensor Decompositions

A Note on Notation A scalar is denoted by a lowercase, italic letter, e.g. x. A column vector is denoted by a lowercase, boldface letter, e.g. $\mathbf{x}$. A matrix is denoted by an uppercase, boldface letter, e.g. $\mathbf{X}$. A 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.

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



(a) Port Scanning Attack-like behaviour on the LBNL Network Traffic Dataset



(b) Over 90% sparser results than regular Parafac, with same approximation error

Fig. 1. A snapshot of our results. In (a) we have one source (addr. 29571) contacts one destination (addr. 30483) using a wide range of near consecutive ports and same amount of packets. In (b), we see that with same relative error, we achieve 90 % sparser outputs, compared to the ALS-Parafac algorithm

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 [14].
The PARAFAC decomposition The Parafac decomposition [12] 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 $\mathbf{a} \circ \mathbf{b} \circ \mathbf{c}(i,j,k) = \mathbf{a}(i)\mathbf{b}(j)\mathbf{c}(k)$.

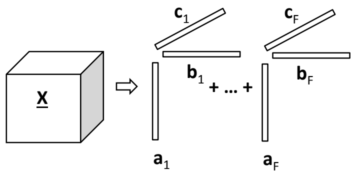


Fig. 2. The F-component Parafac decomposition of $\underline{\mathbf{X}}$.

The most popular algorithm for fitting the Parafac decomposition is the Alternating Least Squares (ALS) [9,14]. The computational complexity of the ALS Algorithm for a $I \times J \times K$ tensor, and for F components is $O(IJKF)$ per iteration.

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.

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 [3]. 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.

[3] 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.

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

The ParCube method defines a means of sampling the tensor across all three modes, without relying on a specific decomposition or a model. Therefore, it satisfies requirement **R3**. Algorithm 1 provides an outline of the sampling forParCube.

Lemma 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 $O(NNZ(\underline{\mathbf{X}}))$.$\blacksquare$

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.

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

```
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 $\underline{\mathbf{X}}_s = \underline{\mathbf{X}}(\mathcal{I}, \mathcal{J}, \mathcal{K})$.
```

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 $\mathcal{I}_p, \mathcal{J}_p, \mathcal{K}_p$ (which is a p fraction of the whole sample). Having this common basis, we are able to combine the samples using Algorithm 4. The basic idea of Algorithm 4 is the following: We arbitrarily choose the factors of one sample to serve as reference, and we distribute their coefficients to the corresponding indices of the factor matrices of the original tensor, as in Algorithm 2. We then process each of the remaining samples individually. For each one of them, we establish a correspondence of the sampled factors to the reference factors, and we update the zero coefficients of the reference factors using values from the current sample. 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.

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 1 provides a first correctness result for our merging algorithm.

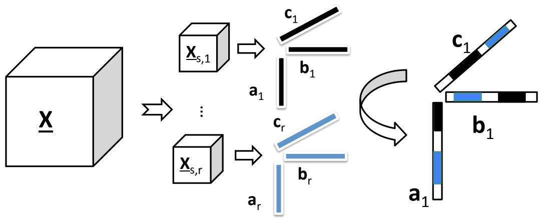


Fig. 3. Example of rank-1 Parafac using ParCube (Algorithm 3). The procedure described is the following: Create r independent samples of $\underline{\mathbf{X}}$, using Algorithm 1. Run the Parafac- ALS algorithm for $K = 1$ and obtain r triplets of vectors, corresponding to the first component of $\underline{\mathbf{X}}$.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

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 $\ell_2$ norm of the columns of the common part:
       $\mathbf{n}_a(f) = \|\mathbf{A}_i(\mathcal{I}_p, f)\|_2$, $\mathbf{n}_b(f) = \|\mathbf{B}_i(\mathcal{J}_p, f)\|_2$, $\mathbf{n}_c(f) = \|\mathbf{C}_i(\mathcal{K}_p, f)\|_2$ for
       $f = 1 \cdots F$. Normalize columns of $\mathbf{A}_i, \mathbf{B}_i, \mathbf{C}_i$ using $\mathbf{n}_a, \mathbf{n}_b, \mathbf{n}_c$ and set
       $\boldsymbol{\lambda}_i(f) = \mathbf{n}_a(f)\mathbf{n}_b(f)\mathbf{n}_c(f)$. Note that the common part will now be normalized
       to unit norm.
6: end for
7: $\mathbf{A}$ =FactorMerge ($\mathbf{A}_i$), $\mathbf{B}$ =FactorMerge ($\mathbf{B}_i$),$\mathbf{C}$ =FactorMerge ($\mathbf{C}_i$)
8: $\boldsymbol{\lambda}$ = 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 $I \times F$.
 1: Set $\mathbf{A} = \mathbf{A}_1$
 2: for $i = 2 \cdots r$ do
 3:     for $f_1 = 1 \cdots F$ do
 4:         for $f_2 = 1 \cdots F$ do
 5:             Compute similarity $\mathbf{v}(f_2) = (\mathbf{A}(\mathcal{I}_p, f_2))^T (\mathbf{A}_i(\mathcal{I}_p, f_1)))$
 6:         end for
 7:         $c = \arg\max_{c'} \mathbf{v}(c')$
 8:         Update only the zero entries of $\mathbf{A}(:, c)$ using vector $\mathbf{A}_i(:, f_1)$.
 9:     end for
10: end for
```

Proposition 1. *Let $(\mathbf{A}, \mathbf{B}, \mathbf{C})$ be the* Parafac *decomposition of $\underline{\mathbf{X}}$, and assume that $\mathbf{A}(\mathcal{I}_p, :)$ ($\mathbf{A}$ restricted to the common I-mode reference rows) is such that any two of its columns are linearly independent; and likewise for $\mathbf{B}(\mathcal{J}_p, :)$ and $\mathbf{C}(\mathcal{K}_p, :)$. Note that if $\mathbf{A}(\mathcal{I}_p, :)$ has as few as 2 rows ($|\mathcal{I}_p| \geq 2$) and is drawn from a jointly continuous distribution, this requirement on $\mathbf{A}(\mathcal{I}_p, :)$ is satisfied with probability 1. Further assume that each of the sub-sampled models is identifiable, and the true underlying rank-one (punctured) factors are recovered, up to permutation and scaling, from each sub-sampled dataset. Then Algorithm 4 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 sketch 1 *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. Those experiments also suggest that increasing the number of samples, r, reduces the Parafac approximation error.

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 [8]. 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 [19], which is the state of the art for Parafac decompositions with sparsity on the latent factors. The speedups we are reporting were measured on a 2.7 GHz Intel Core i5 with 4GB of RAM. Finally, we apply our approach in order to analyze real datasets presented in Table 1.

Name	**Description**	**Dimensions**	**NNZ**
Enron [1]	(sender, recipient, month)	$186 \times 186 \times 44$	9838
Lbnl [18]	(src, dst, port #)	$65170 \times 65170 \times 65327$	27269
Facebook [25]	(wall owner, poster, day)	$63891 \times 63890 \times 1847$	737778
Nell [2]	(noun-phrase, noun-phrase, context)	$14545 \times 14545 \times 28818$	76879419

Table 1. Datasets analyzed

We implemented ParCube in Matlab, and we make it available through the first author's web site [4]. We furthermore use the Tensor Toolbox for Matlab [8] as our core Parafac decomposition implementation.

4.1 Performance & Scalability Evaluation

In the following lines, we evaluate the performance of Parafac using ParCube (Algorithm 2). As a performance metric, we use the 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. In Fig. 4, we measure the relative cost as a function of the speedup incurred by using our ParCube, for different values of the sampling factor; this experiment was carried out on $100 \times 100 \times 100$ randomly generated, synthetic tensors, as we required full control over the true number of components and the degree of sparsity for each component. In Fig. 5, we show the relative cost using the Enron dataset, for various numbers of repetitions (i.e. distinct samples). We see, in this case, that

[4] Download ParCube at `www.cs.cmu.edu/~epapalex/src/parCube.zip`

as the number of repetitions increases, the approximation improves, as expected, from our theoretical result.

4.2 Factor Sparsity Assessment

In Fig.6, we measure the relative output size (i.e. the relative degree of sparsity) between ParCube and Tensor Toolbox non-negative Parafac. 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. In Fig. 7 we measure the relative output size between ParCube and Parafac SLF, as a function of the sampling factor s, for different values of the sparsifying parameter λ used by Parafac SLF (more details in [19]).

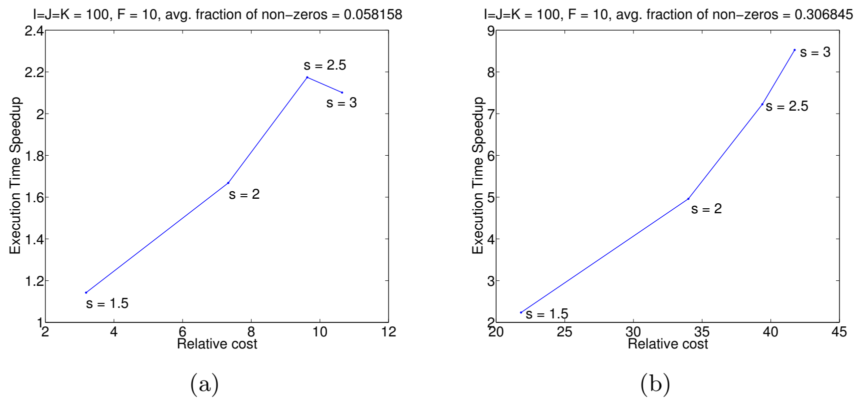
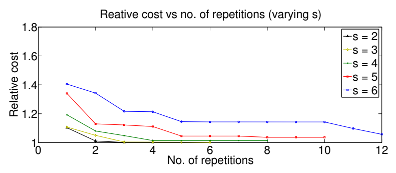


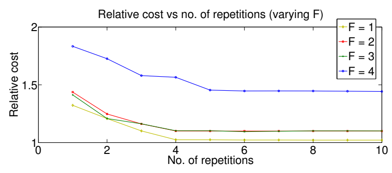
Fig. 4. ParCube is faster than ALS-Parafac: Speedup vs Relative cost (ParCube/ ALS-Parafac) for 1 repetition, for varying sampling factor and different degrees of sparsity. We observe that even for a relatively high sampling factor, we get relatively good relative cost, which may be further improved using repetition. Key here is that by using repetition, because this procedure may be carried out in parallel, we may improve the accuracy and maintain similar speedup.

4.3 Parallelizability

As we have mentioned above, lines 3 to 6 of Algorithm 3 may be carried out entirely in parallel; we have also established, in the previous subsection, that by doing more repetitions, we largely improve the approximation error, and in particular, we converge to the approximation error of the ALS-Parafac algorithm. In its current version, ParCube is not implemented to run in multiple machines, however, here we discuss the potential merits of such an implementation. For evaluation purposes, we divided the run time of Algorithm 3 to a

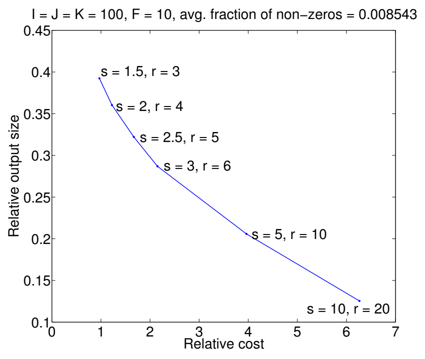


(a) Enron: Relative cost vs No. of repetitons (varying s)

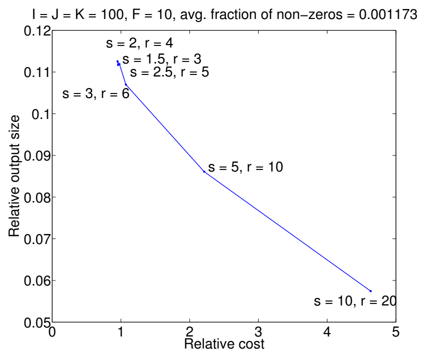


(b) Enron: Relative cost vs No. of repetitons (varying F)

Fig. 5. 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



(a)



(b)

Fig. 6. ParCube outputs sparse factors: Relative Output size (ParCube/ ALS-Parafac) vs Relative cost. We see that the results of ParCube are more than 90% sparser than the ones from Tensor Toolbox, while maintaining the same approximation error.

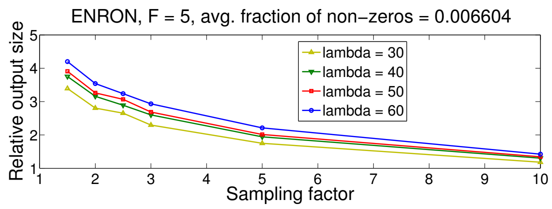


Fig. 7. ParCube outputs sparse factors: Relative Output size (ParCube/ Parafac SLF) vs sampling factor s (where no. of repetitions is $r = 2s$.

parallelizable (lines 3 to 6) and a serial part (everything else). For r repetitions, we added the serial run-time with the *maximum* of the r different, parallel running times. This way, we are loosely emulating the run-time that we would get if we used r cores or machines to run the Algorithm. In particular, by conducting experiments on $256 \times 256 \times 256$ tensors with 0.0578 fraction of non-zeros on average, we got 1.2 average speedup with relative error 1.67 (for $s = 2, r = 4$) and 14.2 speedup with relative error 5.9 (for $s = 10, r = 20$).

4.4 ParCube at work

In this section we present interesting patterns and anomalies, that we were able to discover in the datasets of Table 1, demonstrating that our proposed algorithm ParCube is both practical and effective for data mining practitioners. So far, we don't have an automated method for the selection of parameters s, r, and p, but we leave this for future work; the choice is now made empirically.
ENRON This very well known dataset contains records for 44 months (between 1998 and 2002) of the number of emails exchanged between the 184 employees of the company, forming a $184 \times 184 \times 44$ 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 [7,19]. In Figure 8 we illustrate the temporal evolution of the 4 most prevailing groups in our analysis, having annotated the figure with important events, corresponding to peaks in the communication activity. Labelling of the groups was done manually; because the factors were not very sparse we filtered out very low values on each factor. This issue most certainly stems from the fact that this dataset is not particularly large and therefore by applying the regular ALS-Parafac algorithm to the samples (which is known to yield dense factors), we end up with dense sample factors, which eventually, due to repetition, tend to cover most of the data points. This, however, was not the case for larger datasets analyzed in the following lines, for which the factors turned out to be extremely sparse.

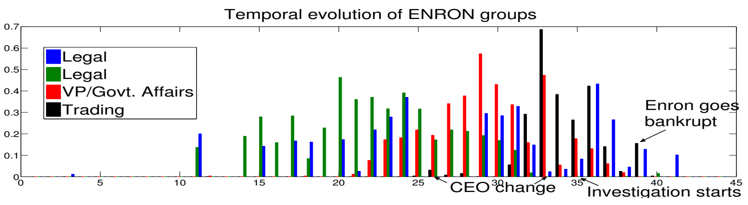


Fig. 8. Temporal evolution of 4 groups in the Enron dataset. We have labelled the groups, according to the position of the participants in the company. The labels of the extracted groups are consistent with other works in the literature [7,19], albeit they have been extracted with somewhat different 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.

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. 9 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.

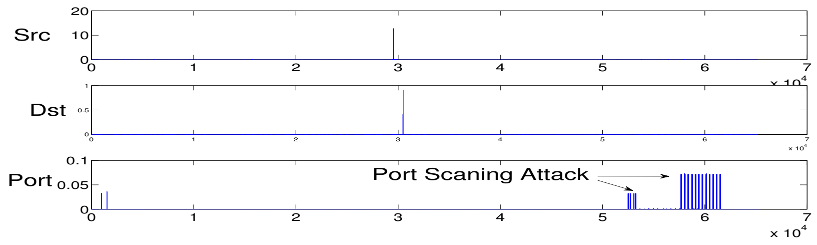
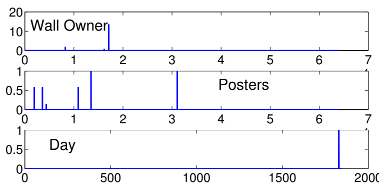


Fig. 9. Anomaly on the LBNL data: We have one source address (addr. 29571), contacting one destination address (addr. 30483) using a wide range of near-consecutive ports, possibly indicating a port scanning attack.

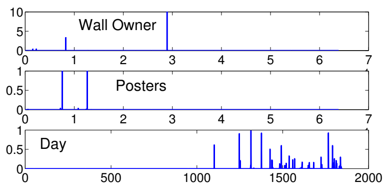
Facebook Wall posts This dataset [5] first appeared in [25]; the specific part of the dataset we used consists of triplets of the form (Wall owner, Poster, day), where the Poster created a post on the Wall owner's Wall on the specified timestamp. By choosing daily granularity, we formed a $63891 \times 63890 \times 1847$ tensor, comprised of 737778 non-zero entries; subsequently, we ran Algorithm 3 using $s = 100$ and $r = 10$. In Figure 10 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.

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. After computing the PARAFAC decomposition of the tensor using PARCUBE with $s = 500$, and $r = 10$ repetitions, we computed the noun-phrase similarity matrix $\mathbf{AA}^T + \mathbf{BB}^T$ and

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



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



(b) Facebook normal activity

Fig. 10. 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

out of that, we were able to discover potential synonyms to noun-phrases, that we report on Table 2.

Noun-phrase	Potential Synonyms
computer	development
period	day, life
months	life
facilities	families, people, communities
rooms	facilities
legs	people
communities	facilities, families, students

Table 2. Nell: Potential synonym discovery

5 Related Work

Tensor applications Tensors and tensor decompositions have gained increasing popularity in the last few years, in the data mining community [14]. The list of tensor applications in data mining is long, however we single out a few that we deemed representative: In [13], the authors extend the well known link analysis algorithm HITS, incorporating textual/topical information. In [7] and [6] the authors use tensors for social network analysis on the Enron dataset. In [22], 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 [10] [16] [3]. Apart from Data Mining, tensors have been and are still being applied in a multitude of fields such as Chemometrics [9] and Signal Processing [21].

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[5,8] (specializing in sparse tensors) and the N-Way Toolbox for Matlab [4] (specializing in dense tensors).
Relevant approaches In [20], the authors propose a partition-and-merge scheme for the PARAFAC decomposition which, however, does not offer factor sparsity. In [19], the authors introduce a PARAFAC decomposition with latent factor sparsity. In [17] and [23] 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, [26] introduces a parallel Non-negative Tensor Factorization. Finally, [24,22] propose randomized, sampling based Tucker3 decompositions.

6 Conclusion

In this work we have introduced PARCUBE, a new, 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, the speedup benefits of PARCUBE may be further improved if we exploit its massive parallelizability. Finally, the practicality of PARCUBE is heavily pronounced by analyzing four different real datasets, discovering patterns and anomalies.

References

1. Enron e-mail dataset. `http://www.cs.cmu.edu/~enron/`.
2. Read the web. `http://rtw.ml.cmu.edu/rtw/`.
3. E. Acar, C. Aykut-Bingol, H. Bingol, R. Bro, and B. Yener. Multiway analysis of epilepsy tensors. *Bioinformatics*, 23(13):i10–i18, 2007.
4. C.A. Andersson and R. Bro. The n-way toolbox for matlab. *Chemometrics and Intelligent Laboratory Systems*, 52(1):1–4, 2000.
5. 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.
6. B.W. Bader, M.W. Berry, and M. Browne. Discussion tracking in enron email using parafac. *Survey of Text Mining II*, pages 147–163, 2008.
7. 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.
8. B.W. Bader and T.G. Kolda. Matlab tensor toolbox version 2.2. *Albuquerque, NM, USA: Sandia National Laboratories*, 2007.
9. R. Bro. Parafac. tutorial and applications. *Chemometrics and intelligent laboratory systems*, 38(2):149–171, 1997.

10. P.A. Chew, B.W. Bader, T.G. Kolda, and A. Abdelali. Cross-language information retrieval using parafac2. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 143–152. ACM, 2007.
11. P. Drineas, R. Kannan, M.W. Mahoney, et al. Fast monte carlo algorithms for matrices iii: Computing a compressed approximate matrix decomposition. *SIAM Journal on Computing*, 36(1):184, 2006.
12. R.A. Harshman. Foundations of the parafac procedure: Models and conditions for an" explanatory" multimodal factor analysis. 1970.
13. 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.
14. T.G. Kolda and B.W. Bader. Tensor decompositions and applications. *SIAM review*, 51(3), 2009.
15. M.W. Mahoney, M. Maggioni, and P. Drineas. Tensor-cur decompositions for tensor-based data. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 327–336. ACM, 2006.
16. 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.
17. D. Nion and N.D. Sidiropoulos. Adaptive algorithms to track the parafac decomposition of a third-order tensor. *Signal Processing, IEEE Transactions on*, 57(6):2299–2310, 2009.
18. R. Pang, M. Allman, M. Bennett, J. Lee, V. Paxson, and B. Tierney. A first look at modern enterprise traffic. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, pages 2–2. USENIX Association, 2005.
19. E.E. Papalexakis and N.D. Sidiropoulos. Co-clustering as multilinear decomposition with sparse latent factors. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 2064–2067. IEEE, 2011.
20. 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.
21. N.D. Sidiropoulos, G.B. Giannakis, and R. Bro. Blind parafac receivers for ds-cdma systems. *Signal Processing, IEEE Transactions on*, 48(3):810–823, 2000.
22. 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.
23. J. Sun, D. Tao, and C. Faloutsos. Beyond streams and graphs: dynamic tensor analysis. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 374–383. ACM, 2006.
24. C.E. Tsourakakis. Mach: Fast randomized tensor decompositions. *Arxiv preprint arXiv:0909.4969*, 2009.
25. Bimal Viswanath, Alan Mislove, Meeyoung Cha, and Krishna P. Gummadi. On the evolution of user interaction in facebook. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Social Networks (WOSN'09)*, August 2009.
26. 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.