HHS Public Access

Author manuscript

Stat Anal Data Min. Author manuscript; available in PMC 2017 August 01.

Published in final edited form as:

Stat Anal Data Min. 2016 August ; 9(4): 269–290. doi:10.1002/sam.11315.

Turbo-SMT: Parallel Coupled Sparse Matrix-Tensor Factorizations and Applications

Evangelos E. Papalexakis[*], **Christos Faloutsos**[*], **Tom M. Mitchell**[*], **Partha Pratim Talukdar**[‡], **Nicholas D. Sidiropoulos**[†], and **Brian Murphy**[§]

Evangelos E. Papalexakis: epapalex@cs.cmu.edu; Christos Faloutsos: christos@cs.cmu.edu; Tom M. Mitchell: tom.mitchell@cmu.edu; Partha Pratim Talukdar: ppt@serc.iisc.in; Nicholas D. Sidiropoulos: nikos@ece.umn.edu; Brian Murphy: brian.murphy@qub.ac.uk

[*]Carnegie Mellon University

[†]University of Minnesota

[‡]Indian Institute of Science (IISc), Bangalore

[§]Queen's University of Belfast

Abstract

How can we correlate the neural activity in the human brain as it responds to typed words, with properties of these terms (like 'edible', 'fits in hand')? In short, we want to find latent variables, that jointly explain both the brain activity, as well as the behavioral responses. This is one of many settings of the *Coupled Matrix-Tensor Factorization* (CMTF) problem.

Can we enhance *any* CMTF solver, so that it can operate on potentially very large datasets that may not fit in main memory? We introduce Turbo-SMT, a meta-method capable of doing exactly that: it boosts the performance of *any* CMTF algorithm, produces sparse and interpretable solutions, and parallelizes *any* CMTF algorithm, producing sparse and interpretable solutions (up to *65 fold*). Additionally, we improve upon ALS, the work-horse algorithm for CMTF, with respect to efficiency and robustness to missing values.

We apply Turbo-SMT to BrainQ, a dataset consisting of a (nouns, brain voxels, human subjects) tensor and a (nouns, properties) matrix, with coupling along the nouns dimension. Turbo-SMT is able to find meaningful latent variables, as well as to predict brain activity with competitive accuracy. Finally, we demonstrate the generality of Turbo-SMT, by applying it on a Facebook dataset (users, 'friends', wall-postings); there, Turbo-SMT spots spammer-like anomalies.

1 Introduction

How is knowledge mapped and stored in the human brain? How is it expressed by people answering simple questions about specific words? If we have data from both worlds, are we able to combine them and jointly analyze them? In a very different scenario, suppose we have the social network graph of an online social network, and we also have additional information about how and when users interacted with each other. What is a comprehensive way to combine those two pieces of data? Both, seemingly different, problems may be

viewed as instances of what is called *Coupled Matrix-Tensor Factorization* (CMTF), where a data tensor and matrices that hold additional information are jointly decomposed into a set of low-rank factors.

The CMTF framework is by no means a new concept, with one of its earliest applications dating back to 2007 where the authors of [11] define a clustering algorithm on data that form Matrix/Tensor couples and apply it to movie recommendation and newsgroup articles settings. Subsequently, there has been a lot of work, both on the algorithmic side [52, 4] and on the application side, ranging from metabolomics [7] to social network analysis [33], where usually the data were of small to medium scale. However, we envision applications of the framework where the data scale renders the current state of the art a bottleneck to the analysis. Such an example of application is the one of *Neurosemantics*, where people are shown stimuli (which can range from single words to entire books) and their brain activity is recorded. These measurements can easily span multiple gigabytes of data, and their size and grows as the complexity of the stimulus increases. To that end, we need fast and parallelizable methods for CMTF that are able to handle data that may not fit in the main memory of a machine. In this work, we introduce Turbo-SMT, a parallel, scalable, and sparsity promoting CMTF meta-algorithm. Our main contributions are the following:

- *Parallel & triple-sparse algorithm:* We provide an approximate, novel, scalable, and triple-sparse (see Sec. 3) meta-method, Turbo-SMT, that is able to parallelize and scale-up *any* CMTF core algorithm. Additionally, we improve upon the ALS core CMTF algorithm, making it faster and able to handle missing values (see Sec. 3 for details on the algorithm, and Sec. 7 for experimental evaluation).
- *Effectiveness & Knowledge Discovery:* As a first step of the Neurosemantics application, we analyze BrainQ, a brain scan dataset which is coupled to a semantic matrix. Furthermore, we analyze a social interaction dataset, coupled with friendship information between the users. (see Sec. 5 for details)
- *Reproducibility:* Our code is publicly available[1] ; the BrainQ and Facebook datasets we use (see Section 5) are also publicly available.

In the knowledge discovery part, the brain scan part of the dataset consists of fMRI scans originally used in [37], a work that first demonstrated that brain activity can be predictably analyzed into component semantic features. Here, we demonstrate a disciplined way to combine both datasets and carry out a variety of data mining/machine learning tasks, through this joint analysis. A snippet of our analysis is shown in Fig. 1, where we find groups of semantically similar nouns that activate the same brain regions; additionally, Turbo-SMT generally improves the CMTF approximation error, while parallelizing the execution of the baselines, working on smaller pieces of data, potentially enabling processing of datasets that may not fit in memory. Furthermore, we apply Turbo-SMT to a time-evolving social network with side information, discovering anomalies, thereby

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

demonstrating the *generality* of the CMTF framework and, ultimately, the strength of Turbo-SMT.

An earlier version of this work was uploaded on Arxiv.org [44] for quick dissemination thereof and early feedback. The first published version of this work is included in the proceedings of SIAM SDM 2014 [45]. The present manuscript is an invited journal extension of [45] as part of the "best of SDM 2014" papers. The new contributions introduced here are 1) additional and extensive performance evaluation, 2) comparison of the Turbo-SMT against compression based techniques (which, to the best of our knowledge have not been published for the CMTF problem), 3) further improvement of the ALS algorithm for CMTF (which is one of the core solvers that can be used with Turbo-SMT), and 4) application of Turbo-SMT to a social network with side information.

2 Preliminaries

2.1 A note on notation

Table 1 contains symbols and essential notation that is henceforth used throughout the text, both in the present section, as well as in Section 3, where Turbo-SMT is introduced.

2.2 Introduction to Tensors

Matrices record dyadic properties, like "people recommending products". Tensors are the n-mode generalizations, capturing 3- and higher-way relationships. For example "subject-verb-object" relationships, such as the ones recorded by the Read the Web - NELL project [1] (and have been recently used in this context [26] [41]) naturally lead to a 3-mode tensor. In this work, our working example of a tensor has three modes. The first mode contains a number of nouns; the second mode corresponds to the brain activity, as recorded by an fMRI machine; and the third mode identifies the human subject corresponding to a particular brain activity measurement. A very popular tensor decomposition, shown in Fig. 2 is the PARAFAC decomposition [24], where a tensor is decomposed into a sum of rank-one tensors (each rank-one tensor being an outer product of three vectors). For a thorough overview of all different flavors of tensor decompositions, we refer the interested reader to [31]

In [41], the authors introduced a scalable and parallelizable tensor decomposition which uses biased mode sampling. In this work, we focus on a more general and expressive framework, that of *Coupled Matrix-Tensor Factorizations*, extending the idea of [41] for that setting. Besides [41], there is a line of work that employs sampling for tensor decomposition [36, 23], which is tangentially related to the present work, and is discussed in more detail in Section 8.3

2.3 Coupled Matrix-Tensor Factorization

Oftentimes, two tensors, or a matrix and a tensor, may have one mode in common; consider the example that we mentioned earlier, where we have a word by brain activity by human subject tensor, we also have a semantic matrix that provides additional information for the same set of words. In this case, we say that the matrix and the tensor are *coupled* in

the 'word' mode, and the problem is instance of Coupled Matrix-Tensor Factorization (CMTF) [11, 52, 4] which is an active field of research, aiming to jointly analyze datasets (in the form of tensors and matrices) that share a subset of their modes.

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

$$\min_{\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{D},\mathbf{E},\mathbf{G}} \left\| \underline{\mathbf{X}} - \sum_k \mathbf{a}_k \circ \mathbf{b}_k \circ \mathbf{c}_k \right\|_F^2 + \|\boldsymbol{Y}_1 - \mathbf{A}\mathbf{D}^T\|_F^2 + \|\boldsymbol{Y}_2 - \mathbf{B}\mathbf{E}^T\|_F^2 + \|\boldsymbol{Y}_3 - \mathbf{C}\mathbf{G}^T\|_F^2 \tag{2.1}$$

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

2.4 Solving CMTF

A popular algorithm for solving CMTF (as shown in Figure 3) is the so-called Alternating Least Squares (ALS), a block-coordinate descent method [4] (Fig. 5); in the next subsection, we provide a short description of the ALS algorithm, since one of our contributions pertains to speeding up the ALS algorithm itself. Besides ALS, there exist other algorithms for CMTF. For example, [4] uses a first order optimization algorithm for the same objective. Throughout this work, we use both algorithms as core CMTF solvers for Turbo-SMT. The strength of Turbo-SMT, however, is that it can be used as-is with any underlying core CMTF implementation. Hence, Turbo-SMT is *CMTF solver independent*.

2.4.1 The Alternating Least Squares Algorithm—The work-horse algorithm for solving PARAFAC (as shown in Figure 2) is Alternating Least Squares (ALS) [50]; the basic idea is that by fixing two of the three factor matrices, we have a least squares problem for the third, and we thus do so iteratively, alternating between the matrices we fix and the one we optimize for, until the algorithm converges, usually when the relative change in the objective function between two iterations is very small.

Solving CMTF using ALS follows the same strategy, only now, we have up to three additional matrices in our objective. For instance, when fixing all matrices but $\mathbf{A}$, the update for $\mathbf{A}$ requires to solve the following least squares problem:

$$\min_{\mathbf{A}} \|\underline{\mathbf{X}}_{(1)} - (\mathbf{B} \odot \mathbf{C})\mathbf{A}^T\|_F^2 + \|\boldsymbol{Y}_1 - \mathbf{D}\mathbf{A}^T\|_F^2$$

In order to obtain initial estimates for matrices **A, B, C** we take the PARAFAC decomposition of $\underline{\mathbf{X}}$. As for matrix **D** (and similarly for the rest), it suffices to solve a simple Least Squares problem, given the PARAFAC estimate of **A**, we initialize as $\mathbf{D} = \mathbf{Y}_1 (\mathbf{A}^{\dagger})^T$, where † denotes the Moore-Penrose Pseudoinverse which, given the Singular Value Decomposition of a matrix $\mathbf{X} = \mathbf{U\Sigma V}^T$, is computed as $\mathbf{X}^{\dagger} = \mathbf{V\Sigma^{-1}U}^T$.

In Algorithm 1, we provide a detailed outline of the baseline algorithm, ALS for CMTF, also shown in Figure 5 of [4] in less detail.

The Moore-Penrose Pseudoinverse of a matrix is computed as $\mathbf{X}^{\dagger} = \mathbf{V\Sigma^{-1}U}^T$ given the Singular Value Decomposition of a matrix $\mathbf{X} = \mathbf{U\Sigma V}^T$. The Kronecker product of two matrices **A, B** (of sizes $I \times J$ and $K \times M$ respectively) is defined as

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

Algorithm 1

Baseline: Alternating Least Squares (ALS) Algorithm for CMTF [4]

Input: $\underline{\mathbf{X}}$ of size $I \times J \times K$, matrices $\mathbf{Y}_i$, $i = 1 \cdots 3$, of size $I \times I_2$, $J \times J_2$, and $K \times K_2$ respectively, number of factors F.

Output: **A** of size $I \times F$, **B** of size $J \times F$, **C** of size $K \times F$, **D** of size $I_2 \times F$, **G** of size $J_2 \times F$, **E** of size $K_2 \times F$.

1: Unfold $\underline{\mathbf{X}}$ into $\mathbf{X}_{(1)}$, $\mathbf{X}_{(2)}$, $\mathbf{X}_{(3)}$ (see [27]).

2: Initialize **A**, **B**, **C** using PARAFAC of $\underline{\mathbf{X}}$. Initialize **D**, **G**, **E** as discussed on the text (e.g. $\mathbf{D} = \mathbf{Y}_1 (\mathbf{A}^{\dagger})^T$).

3: **while** convergence criterion is not met **do**

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

5: $$\mathbf{B} = \begin{bmatrix} \mathbf{X}_{(2)} \\ \boldsymbol{Y}_2 \end{bmatrix}^T \left(\begin{bmatrix} \mathbf{C} \odot \mathbf{A} \\ \mathbf{G} \end{bmatrix}^{\dagger} \right)^T$$

6: $$\mathbf{C} = \begin{bmatrix} \mathbf{X}_{(3)} \\ \boldsymbol{Y}_3 \end{bmatrix}^T \left(\begin{bmatrix} \mathbf{A} \odot \mathbf{B} \\ \mathbf{E} \end{bmatrix}^{\dagger} \right)^T$$

7: $\mathbf{D} = \mathbf{Y}_1 (\mathbf{A}^{\dagger})^T$, $\mathbf{G} = \mathbf{Y}_2 (\mathbf{B}^{\dagger})^T$, $\mathbf{E} = \mathbf{Y}_3 (\mathbf{C}^{\dagger})^T$

8: **end while**

The Khatri-Rao product of two matrices **A, B** with sizes $I \times F$ and $J \times F$ is

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

In Section 4.1, we improve upon Algorithm 1, introducing a few key modifications that allow for faster execution.

3 Proposed Method

3.1 Algorithm Description

There are three main concepts behind Turbo-SMT (outlined in Algorithm 2):

Phase 1 Obtain a *nucleus* of our data by using biased sampling.

Phase 2 Fit CMTF to the reduced data (possibly on more than one *nuclei*)

Phase 3 stitch the partial results

Phase 1: Sampling—An efficient way to reduce the size of the dataset, yet operate on a representative subset thereof is to use *biased* sampling. In particular, given a three-mode tensor $\underline{\mathbf{X}}$ we sample as follows. We calculate three vectors as shown in equations (3.2), one for each mode of $\underline{\mathbf{X}}$ (and respectively Equations 3.3 and 3.4 for the two modes of the matrix).

$$\begin{aligned}\mathbf{x}_A(i) &= \sum_{j=1}^{J}\sum_{k=1}^{K}|\underline{\mathbf{X}}(i,j,k)| \\ &+ \sum_{j=1}^{I_1}|\boldsymbol{Y}_1(i,j)|, \mathbf{x}_B(j) \\ &= \sum_{i=1}^{I}\sum_{k=1}^{K}|\underline{\mathbf{X}}(i,j,k)| \\ &+ \sum_{i=1}^{I_2}|\boldsymbol{Y}_2(j,i)|, \mathbf{x}_C(k) \\ &= \sum_{i=1}^{I}\sum_{j=1}^{J}|\underline{\mathbf{X}}(i,j,k)| \\ &+ \sum_{j=1}^{I_3}|\boldsymbol{Y}_3(k,j)|, \end{aligned} \tag{3.2}$$

$$\mathbf{y}_{1,A}(i) = \sum_{j=1}^{I_1}|\boldsymbol{Y}_1(i,j)| \mathbf{y}_{2,B}(j) = \sum_{i=1}^{I_2}|\boldsymbol{Y}_2(j,i)|, \mathbf{y}_{3,C}(k) = \sum_{j=1}^{I_3}|\boldsymbol{Y}_3(k,j)| \tag{3.3}$$

$$\mathbf{y}_{1,D}(j) = \sum_{i=1}^{I}|\boldsymbol{Y}_1(i,j)|, \mathbf{y}_{2,G}(i) = \sum_{j=1}^{J}|\boldsymbol{Y}_2(j,i)|, \mathbf{y}_{3,E}(i) = \sum_{k=1}^{K}|\boldsymbol{Y}_3(k,i)| \tag{3.4}$$

These vectors, which we henceforth refer to as *density vectors* are the marginal absolute sums with respect to all but one of the modes of the tensor, and in essence represent the importance of each index of the respective mode. We then sample *indices* of each mode according to the respective density vector. For instance, assume an $I \times J \times K$ tensor; suppose

that we need a sample of size $\frac{I}{s}$ of the indices of the first mode. Then, we just define $p_{\mathcal{I}}(i)=\mathbf{x}_A(i)/\sum_{i=1}^{I}\mathbf{x}_A(i)$ as the probability of sampling the i-th index of the first mode, and we simply sample without replacement from the set $\{1 \cdots I\}$, using $p_{\mathcal{I}}$ as bias. The very same idea is used for matrices $\mathbf{Y}_i$. Doing so is preferable over sampling uniformly, since our bias makes it more probable that high density indices of the data will be retained on the sample, and hence, it will be more representative of the entire set.

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

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

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

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

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

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

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

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

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

In order to make the merging work, we first introduce the following scaling on each column of each factor matrix: Let's take $\mathbf{A}^{(i)}$ for example; we normalize each column of $\mathbf{A}$ by the ℓ_2 norm of the common part, as described in line 8 of Algorithm 2. By doing so, the common part of each factor matrix (for all repetitions) will be unit norm. This scaling is absorbed in a set of scaling vectors $\boldsymbol{\lambda}_A$ (and accordingly for the rest of the factors). The new objective function is shown in Equation 3.5

$$\begin{aligned}\min_{\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{D},\mathbf{E},\mathbf{G}} &\left\|\underline{\mathbf{X}} - \sum_k \boldsymbol{\lambda}_A(k)\boldsymbol{\lambda}_B(k)\boldsymbol{\lambda}_C(k)\mathbf{a}_k \circ \mathbf{b}_k \circ \mathbf{c}_k\right\|_F^2 \\ &+ \|\boldsymbol{Y}_1 - \mathbf{A}\ \mathrm{diag}(\boldsymbol{\lambda}_A * \boldsymbol{\lambda}_D)\mathbf{D}^T\|_F^2 \\ &+ \|\boldsymbol{Y}_2 - \mathbf{B}\ \mathrm{diag}(\boldsymbol{\lambda}_B * \boldsymbol{\lambda}_E)\mathbf{E}^T\|_F^2 \\ &+ \|\boldsymbol{Y}_3 - \mathbf{C}\ \mathrm{diag}(\boldsymbol{\lambda}_C * \boldsymbol{\lambda}_G)\mathbf{G}^T\|_F^2 \end{aligned} \quad (3.5)$$

A problem that is introduced by carrying out multiple sampling repetitions is that the correspondence of the output factors of each repetition is very likely to be distorted. In other

words, say we have matrices $\mathbf{A}^{(1)}$ and $\mathbf{A}^{(2)}$ and we wish to merge their columns (i.e. the latent components) into a single matrix **A**, by stitching together columns that correspond to the same component. It might very well be the case that the order in which the latent components appear in $\mathbf{A}^{(1)}$ is not the same as in $\mathbf{A}^{(2)}$.

The sole purpose of the aforementioned normalization is to resolve the correspondence problem. In Algorithm 3, we merge the partial results while establishing the correct correspondence of the columns.

Algorithm 3 can be seen as a greedy algorithm for solving the correspondence problem between columns. Provided that the factor matrices are not collinear, the algorithm usually finds the correct correspondence. In reality, where data might be noisy, we can use the Hungarian Method [32], which will slightly increase the computational cost of the overall algorithm but solve the problem optimally.

In order to see why this happens, we follow the example of $r = 2$ of the previous paragraph, according to Algorithm 3, we compute the inner product of the common parts of each column of $\mathbf{A}^{(1)}$ and $\mathbf{A}^{(2)}$. Since the common parts of each column are normalized to unit norm, then the inner product of the common part of the column of $\mathbf{A}^{(1)}$ with that of $\mathbf{A}^{(2)}$ will be maximized (and exactly equal to 1) for the matching columns, and by the Cauchy-Schwartz inequality, for all other combinations, it will be less than 1. Additionally, elimination of the already used columns operates as a tie-breaker.

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

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

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

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

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

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

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

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

and

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

which can, again, be solved using the Hungarian Method [32]. However, as we mentioned earlier, the size of $\mathscr{Q}_p$, $\mathscr{J}_p$, $\mathscr{K}_p$, in cases where we are dealing with big tensors and matrices, can be in the orders of thousands or tens of thousands. Thus, computing the above Khatri-Rao products may incur high computational and memory overhead during the stitching step, thus, in Algorithm 2 we show Turbo-SMT when using the lightweight version of stitching (Algorithm 3).

Algorithm 2

Turbo-SMT: Sparse and parallel CMTF

Input: Tensor $\underline{\mathbf{X}}$ of size $I \times J \times K$, matrices $\mathbf{Y}_i$ $i = 1 \cdots 3$, of size $I \times I_2$, $J \times J_2$, and $K \times K_2$ respectively, number of factors F,
sampling factor s, number of repetitions r.

Output: $\mathbf{A}$ of size $I \times F$, $\mathbf{b}$ of size $J \times F$, $\mathbf{c}$ of size $K \times F$, $\mathbf{D}$ of size $I_2 \times F$, $\mathbf{G}$ of size $J_2 \times F$, $\mathbf{E}$ of size $K_2 \times F$. λ_A, λ_B, λ_C, λ_D, λ_E, λ_G of size $F \times 1$.

1: Initialize $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{D}$, $\mathbf{E}$, $\mathbf{G}$ to all-zeros.

2: Randomly, *using mode densities as bias*, select a set of 100p% ($p \in [0, 1]$) "anchor" indices $\mathscr{Q}_p$ to be common across all repetitions.

For example, $\mathscr{Q}_p$ is sampled with probabilities with $p_{\mathscr{I}}(i) = \mathbf{x}_A(i) / \sum_{i=1}^{I} \mathbf{x}_A(i)$.

3: **for** $i = 1 \cdots r$ **do**

{**Phase 1: Obtain *nuclei* through biased sampling**}

4: Compute densities as in equations 3.2, 3.3, 3.4.

Compute set of indices $\mathcal{R}^{(i)}$ as random sample without replacement of $\{1 \cdots I\}$ of size $I/(s(1-p))$ with probability $p_{\mathcal{I}}(i) = \mathbf{x}_A(i) / \sum_{i=1}^{I} \mathbf{x}_a(i)$. Likewise for $\mathcal{J}$, $\mathcal{K}$,, $\mathcal{R}_1$, $\mathcal{R}_2$, and $\mathcal{R}_3$. Set $\mathcal{R}^{(i)} = \mathcal{R} \cup \mathcal{R}_p$.

5: Get *nucleus* $\underline{\mathbf{X}}_s^{(i)} = \underline{\mathbf{X}}(\mathcal{I}^{(i)}, \mathcal{J}^{(i)}, \mathcal{K}^{(i)})$, $\boldsymbol{Y}_{1s}^{(i)} = \boldsymbol{Y}_1(\mathcal{I}^{(i)}, \mathcal{I}_1^{(i)})$ and likewise for $\boldsymbol{Y}_{2s}^{(i)}$ and $\boldsymbol{Y}_{3s}^{(i)}$. Note that the same index sample is used for *coupled* modes.

{**Phase 2: Fit the model on each *nucleus***}

6: Run a CMTF solver for $\mathbf{X}_s^{(i)}$ and $\mathbf{Y}_{js}^{(i)}$, $j = 1 \cdots 3$ and obtain $\mathbf{A}_s$, $\mathbf{B}_s$, $\mathbf{C}_s$, $\mathbf{D}_s$, $\mathbf{G}_s$, $\mathbf{E}_s$.

7: $\mathbf{A}^{(i)}(\mathcal{R}^{(i)}, :) = \mathbf{A}_s$. Likewise for the rest.

8: Calculate the ℓ_2 norm of the columns of the common part: $\boldsymbol{\lambda}_A^{(i)}(f) = \|\mathbf{A}^{(i)}(\mathcal{I}_p, f)\|_2$, for $f = 1 \cdots F$. Normalize columns of $\mathbf{A}^{(i)}$ using $\boldsymbol{\lambda}_A^{(i)}$ (likewise for the rest). Note that the common part of each factor will now be normalized to unit norm.

9: **end for**

{**Phase 3: Stitch partial results**}

10: $\mathbf{A}$ =StitchFactors ($\mathbf{A}_1^i$). When stitching partial results for $\mathbf{A}$, record the column correspondences for different nuclei

11: Using the above correspondence, stitch the partial factors of **B**, **C**, **D**, **E**, **G**, without performing the full StitchFactors algorithm, but simply reordering the columns.

12: Using the above column/component correspondence, reorder λ_A, λ_B, λ_C, λ_D, λ_E, λ_G for each repetition.

13: λ_A = average of λ_{A1}^i. Likewise for the rest.

3.2 Sparsity through Sampling

Besides data size reduction, one merit of sampling is sparsity on the latent factors. Every time Turbo-SMT does one repetition, it operates on a sub-sampled version of the data. Consequently, in the third phase of the algorithm, where the results are re-distributed to their indices in the original, high dimensional space, most of the indices of the latent factors are going to be exactly zero, thus resulting in latent factor sparsity. In this way, Turbo-SMT always operates on a sparse set of data, through the entire lifetime of the algorithm, a thing which is not true for the majority of the algorithms both for tensor and coupled decompositions, which usually operate on dense factors (even when the final output is sparse), and have very high and unnecessary storage needs.

Definition 3.1. (Triple-sparse)—*An algorithm is triple-sparse when 1) the input of the algorithm is sparse, 2) the intermediate data during the lifetime of the algorithm is sparse, and 3) the final output of the algorithm is sparse*.

In the above definition, the input of the algorithm need not necessarily be sparse; however, a triple-sparse algorithm still satisfies the second and third requirement, by operating on a

sparse, representative subset of the data. We, thus, call Turbo-SMT, a *triple-sparse* algorithm. We have to note that we consider the intermediate data that the algorithm manipulates to be sparse *with respect to the original data*, and not necessarily sparse with respect to the sample size.

In addition to storage and efficiency, a major benefit of sparsity is *interpretability*: when most of the coefficients of the factors are zero, it is easier for a person to inspect the non-zeros and identify patterns in the results. Additionally, having sparse latent factors has been shown to be equivalent to higher order co-clustering [42, 43] which is a very useful and highly interpretable tool in exploratory data mining.

Algorithm 3

StitchFactors: Given partial results of factor matrices, merge them correctly

Input: Factor matrices $\mathbf{A}_1^i$ of size $I \times F$ each, and r is the number of repetitions, $\mathcal{Q}_p$: set of common indices.

Output: Factor matrix $\mathbf{A}$ of size $I \times F$.

1: Set $\mathbf{A} = \mathbf{A}^{(1)}$

2: Set $\ell = \{1 \cdots F\}$, a list that keeps track of which columns have not been assigned yet.

3: **for** $i = 2 \cdots r$ **do**

4: **for** $f_1 = 1 \cdots F$ **do**

5: **for** f_2 in ℓ **do**

6: Compute similarity $\mathbf{v}(f_2) = (\mathbf{A}(\mathcal{Q}_p, f_2))^T (\mathbf{A}^{(i)}(\mathcal{Q}_p, f_1))$

7: **end for**

8: $c^* = \arg\max_c \mathbf{v}(c)$ (Ideally, for the matching columns, the inner product should be equal to 1; conversely, for the rest of the columns, it should be considerably smaller)

9: $\mathbf{A}(:, c^*) = \mathbf{A}^{(i)}(:, f_1)|_{\mathbf{A}(:,c^*)=0}$, i.e. update the zero entries of the column.

10: Remove c^* from list ℓ

11: **end for**

12: **end for**

3.3 Parallelization

Turbo-SMT is, by its nature, parallelizable; in essence, we generate multiple samples of the coupled data, we fit a CMTF model to each sample and then we merge the results. By carefully observing Algorithm 2, we can see that lines 3 to 9 may be carried out entirely in parallel, provided that we have a good enough random number generator that does not generate the very same sample across all r repetitions. In particular, the r repetitions are independent from one another, since computing the set of common indices (line 2), which is the common factor across all repetitions, is done before line 3.

4 Further Optimizations

4.1 Speeding up the ALS algorithm

In addition to our main contribution in terms of speeding CMTF in general, we are able to further speed the ALS algorithm up, by making a few careful interventions to the core algorithm (Algorithm 1).

Lemma 4.1—*We may do the following simplification to each pseudoinversion step of the ALS algorithm* (Algorithm 1):

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

Proof: For the Moore-Penrose pseudoinverse of the Khatri-Rao product, it holds that [13], [34]

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

Furthermore [13] $(\mathbf{A} \odot \mathbf{B})^T (\mathbf{A} \odot \mathbf{B}) = \mathbf{A}^T\mathbf{A} * \mathbf{B}^T\mathbf{B}$ For a partitioned matrix $\mathbf{P} = \begin{bmatrix} \mathbf{P}_1 \\ \mathbf{P}_2 \end{bmatrix}$, it holds that its pseudoinverse may be written in the following form [25]

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

Putting things together, it follows:

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

which concludes the proof.

The above lemma implies that substituting the naive pseudoinversion of $\begin{bmatrix} \mathbf{A} \odot \mathbf{B} \\ \mathbf{M} \end{bmatrix}$ with the simplified version, offers significant *computational* gains to Algorithm 1. More precisely, if the dimensions of **A, B** and **M** are $I \times R$, $J \times R$ and $I \times I_2$, then computing the pseudoinverse naively would cost $O(R^2 (IJ + I_2))$, whereas our proposed method yields a cost of $O(R^2 (I + J + I_2))$ because of the fact that we are pseudoinverting only a *small* $R \times R$ matrix. We have to note here that in almost all practical scenarios $R \ll I, J, I_2$.

Table 2 provides a solid impression of the speedup achieved on the core ALS algorithm, as a result of the simplification of the pseudo-inversion step, as derived above. In short, we can

see that the speedup achieved is in most realistic cases 2× or higher, adding up to being a significant improvement on the traditional algorithm.

4.2 Making ALS robust to missing values

In many practical scenarios, we often have corrupted or missing data. For instance, when measuring brain activity, a few sensors might stop working, whereas the majority of the sensors produce useful signal. Despite these common data imperfections, it is important for a data mining algorithm to be able to operate. The work of Tomasi et. al [49] provides a very comprehensive study on how to handle missing values for plain tensor decompositions. A very clean and straightforward way of handling missing values is to *ignore* them throughout the optimization process, both with respect to the original data and with respect to the model. This can be achieved through masking the missing values, hiding them from the optimization. Notice that is *not* the same as simply zeroing out all missing values, since 0 might have a valid physical interpretation.

In this section, we show how this masking can be applied to the ALS algorithm for the CMTF model. In [4], the authors show how this can be achieved for the CMTF-OPT algorithm. In any case, this masking approach is very important because besides enabling the analysis of incomplete datasets, it can also serve as stepping stone for estimating those missing values: When obtaining a low rank model of the incomplete dataset after masking the missing values, we can reconstruct the original data and our reconstruction will contain imputations for the missing values. This can be seen as the first iteration of an Expectation-Maximization scheme introduced in [49].

Following the formulation of [6, 4] we define a 'weight' tensor $\underline{\mathbf{W}}$ which has '0' in all coefficients where values are missing, and '1' everywhere else. Similarly, we introduce three weight matrices $\mathbf{W}_i$ for each of the coupled matrices $\mathbf{Y}_i$. Then, the optimization function of the CMTF model becomes

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

As we show in Algorithm 1, we may solve CMTF by solving six least squares problems in an alternating fashion. A fortuitous implication of this fact is that in order to handle missing values for CMTF, it suffices to solve

$$\min_{\mathbf{B}} \left\| \mathbf{W} * \left(\mathbf{X} - \mathbf{A}\mathbf{B}^T \right) \right\|_F^2 \quad (4.6)$$

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

Our solution draws from a similar derivation in [43] (Section 3.B) On our way tackling the above problem, we first need to investigate its scalar case, i.e. the case where we are

interested only in **B**(*j, f*) for a fixed pair of *j* and *f*. The optimization problem may be rewritten as

$$\min_{\mathbf{B}(j,f)} \|\mathbf{W}(:,j)*\mathbf{X}(:,j) - (\mathbf{W}(:,j)*\mathbf{A}(:f))\mathbf{B}(j,f)^T\| \quad (4.7)$$

which is essentially a scalar least squares problem of the form: $\min_b \|\mathbf{x} - \mathbf{a}b\|_2^2$ with solution in analytical form: $b = \frac{\mathbf{x}^T\mathbf{a}}{\|\mathbf{a}\|_2^2}$. The reader may notice that the dimensions of **W** and **A** are incompatible, however, in the scalar case of Eq. 4.7, we multiply element-wise the columns of **W** and **A**, which are of the same dimensions.

We may, thus, solve this problem of Equation 4.6 using coordinate descent, on **B** iteratively, until convergence. Therefore, with the aforementioned derivation, we are able to modify our original algorithm in order to take missing values into account.

So far, we have described how to make ALS robust against missing values, however a few modifications to Turbo-SMT are required in order to be able to use the robust ALS core solver. In particular, in order to use the masking scheme for Turbo-SMT, during sampling, we have to take into account that some values in the data are missing: when we compute the weights for the biased sampling (i.e. Equations 3.2, 3.3, 3.4) we have to hide the missing values. In order to do that, we compute the weights on

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

and

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

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

5 Knowledge Discovery

5.1 Turbo-SMT on Brain Image Data With Additional Semantic Information

As part of a larger study of neural representations of word meanings in the human brain [37], we applied Turbo-SMT to a combination of datasets which we henceforth jointly refer to as BrainQ. This dataset consists of two parts. The first is a tensor that contains measurements of the fMRI brain activity of 9 human subjects, when shown each of 60 concrete nouns (5 in each of 12 categories, e.g. dog, hand, house, door, shirt, dresser, butterfly, knife, telephone, saw, lettuce, train). fMRI measures slow changes in blood oxygenation levels, reflecting localized changes in brain activity. Here our data is made up

of 3×3×6mm voxels (3D pixels) corresponding to fixed spatial locations across participants. Recorded fMRI values are the mean activity over 4 contiguous seconds, averaged over multiple presentations of each stimulus word (each word is presented 6 times as a stimulus). Further acquisition and preprocessing details are given in [37]. This dataset is publicly available[2]. The second part of the data is a matrix describing the semantics of these 60 nouns, using crowd-sourced numerical responses to 218 questions such as "is it smaller than a golfball?" (see also samples in Fig. 4). This dataset has been used before in works such as [39], [40].

BrainQ's size is 60×77775×9 with over 11 million non-zeros (tensor), and 60 × 218 with about 11000 non-zeros (matrix). The dimensions might not be extremely high, however, the data is *very dense* and it is therefore difficult to handle efficiently.

5.2 Simultaneous Clustering of Words, Questions and Regions of the Brain

One of the strengths of CMTF is its expressiveness in terms of simultaneously soft-clustering all involved entities of the problem. By taking a low rank decomposition of the BrainQ data (using $r = 5$ and $s_I = 3$, $s_J = 86$, $s_K = 1$ for the tensor and s_I for the questions dimension of the matrix)[3], we are able to find groups that jointly express words, questions and brain voxels (we can also derive groups of human subjects; however, it is an active research subject in neuroscience, whether brain-scans should differ significantly between people, and is out of the scope of the present work).

In Figure 4, we display 4 such groups of brain regions that are activated given a stimulus of a group of similar words; we also display the most prominent words, along with groups of similar questions that were highly correlated with the words of each group. Moreover, we were able to successfully identify high activation of the *premotor cortex* in Group 3, which is associated with concepts that have conventional manual uses (such as holding, or picking up).

5.3 Predicting Brain Activity from Questions

In addition to soft-clustering, the low rank joint decomposition of the BrainQ data offers another significant result. This low dimensional embedding of the data into a common semantic space, enables the prediction of, say, the brain activity of a subject, for a given word, given the corresponding vector of question answers for that word. In particular, we use this linear transformation in order to map the question answer vector to the latent semantic space and then expanding it to the brain voxel space, we obtain a fairly good prediction of the brain activity. This linear transformation is reminiscent of the mapping done in Latent Semantic Analysis (LSA) [21], where the authors compute a low rank embedding of terms and documents, and given a query consisting of terms, they project it to the latent dimension, taking advantage of the compaction offered by the low rank approximation. In the case of LSA, the embeddings used are orthonormal (since they are computed using SVD), and thus

[2] http://www.cs.cmu.edu/afs/cs/project/theo-73/www/science2008/data.html
[3] We may use imbalanced sampling factors, especially when the data is far from being 'rectangular'.

the "mapping" is a proper projection to the subspace. In our case, we do not restrict the embeddings to be orthonormal, and thus we perform a linear transformation.

To evaluate the accuracy of these predictions of brain activity, we follow a *leave-two-out* scheme, where we remove two words entirely from the brain tensor and the question matrix; we carry out the joint decomposition, in some very low dimension, for the remaining set of words and we obtain the usual set of matrices **A, B, C, D**. Due to the randomized nature of Turbo-SMT, we did 100 repetitions of the procedure described below.

Let $\mathbf{q}_i$ be the question vector for some word *i*, and $\mathbf{v}_i$ be the brain activity of one human subject, pertaining to the same word. By left-multiplying $\mathbf{q}_i$ with $\mathbf{D}^T$, we map $\mathbf{q}_i$ to the latent space of the decomposition; then, by left-multiplying the result with **B**, we map the result to the brain voxel space. Thus, our estimated (predicted) brain activity is obtained as $\hat{\mathbf{v}}_i = \mathbf{B}\mathbf{D}^T\mathbf{q}_i$

Given the predicted brain activities $\hat{\mathbf{v}}_1$ and $\hat{\mathbf{v}}_2$ for the two left out words, and the two actual brain images $\mathbf{v}_1$ and $\mathbf{v}_2$ which were withheld from the training data, the *leave-two-out* scheme measures prediction accuracy by the ability to choose which of the observed brain images corresponds to which of the two words. After mean-centering the vectors, this classification decision is made according to the following rule:

$$\|\mathbf{v}_1 - \hat{\mathbf{v}}_1\|_2 + \|\mathbf{v}_2 - \hat{\mathbf{v}}_2\|_2 < \|\mathbf{v}_1 - \hat{\mathbf{v}}_2\|_2 + \|\mathbf{v}_2 - \hat{\mathbf{v}}_1\|_2$$

Although our approach is not designed to make predictions, preliminary results are very encouraging: Using only F=2 components, for the noun pair *closet/watch* we obtained mean accuracy of about 0.82 for 5 out of the 9 human subjects. Similarly, for the pair *knife/beetle*, we achieved accuracy of about 0.8 for a somewhat different group of 5 subjects. For the rest of the human subjects, the accuracy is considerably lower, however, it may be the case that brain activity predictability varies between subjects, a fact that requires further investigation.

6 Generality: Mining Social Networks with Additional Information

The CMTF framework, and Turbo-SMT by the same token, can be applied in a very diverse set of applications. In the previous subsection we demonstrated the expressive power of the framework for the BrainQ dataset; here we show how it can be applied in a completely different scenario, that of social network analysis with side information. To that end, we use Turbo-SMT to analyze a Facebook dataset, introduced in [51][4]. This dataset consists of a 63890 × 63890 × 1847 (wall, poster, day) tensor with about 740,000 non-zeros, and a 63890 × 63890 who is friends with whom matrix, with about 1.6 million non-zeros. The tensor contains the amount of posts a user posted on some user's Wall. In principle it contains counts, however, most of the values are either 1 or 0. In contrast to BrainQ, this dataset is very sparse (as one would expect from a social network dataset). However, Turbo-SMT works in both cases, demonstrating that it can analyze data efficiently, regardless of their density.

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

We decomposed the data into 25 rank one components, using $s_I = 1000$, $s_J = 1000$, $s_K = 100$ and s_I for both dimensions of the matrix, and manually inspected the results. A fair amount of components captured normal activity of Facebook users who occasionally post on their friends' walls; here we only show one outstanding anomaly, due to lack of space: In Fig. 5 we show what appears to be a spammer, i.e. a person who, only on a certain day, posts on many different friends' walls: the first subfigure corresponds to the wall owners, the second subfigure corresponds to the people who post on these walls, and the third subfigure is the time (measured in days); we thus have one person, posting on many peoples' walls, on a single day.

We chose to include this particular result in order to showcase the modelling versatility of CMTF models, and thus of the Turbo-SMT algorithm, since we are able to express a variety of problems in the CMTF framework, and additionally, solve them very efficiently, and obtain sparse and interpretable solutions by using Turbo-SMT. The Facebook dataset merits its own detailed analysis, however, here we include a small, preliminary result, due to space considerations.

7 Experiments

7.1 Experimental Setup

We implemented Turbo-SMT in Matlab. Our implementation of the code is publicly available.[5] For the parallelization of the algorithm, we used Matlab's Parallel Computing Toolbox. For tensor manipulation, we used the Tensor Toolbox for Matlab [10] which is optimized especially for sparse tensors (but works very well for dense ones too). We use the CMTF-ALS and the CMTF-OPT [4] algorithms as baselines, i.e. we compare Turbo-SMT when using one of those algorithms as their core CMTF implementation, against the plain execution of those algorithms. In order to make this comparison possible, all datasets we used were within the capabilities of the Tensor Toolbox for Matlab.

We implemented our version of the ALS algorithm, and we used the CMTF Toobox[6] implementation of CMTF-OPT. All experiments were carried out on a machine with 4 Intel Xeon E74850 2.00GHz, and 512Gb of RAM. The version of Matlab was R2013a (8.1.0.604) 64-bit. In the appendix, we show timing experiments on a less powerful machine, in order to demonstrate that the principle behind Turbo-SMT is applicable to a variety of platforms. Whenever we conducted multiple iterations of an experiment (due to the randomized nature of Turbo-SMT), we report error-bars along the plots. For all the following experiments we used either portions of the BrainQ dataset, or the whole dataset.

7.2 Run time

A major feature of Turbo-SMT is scaling up and boosting existing, potentially highly optimized and state of the art, CMTF solvers. Key facts that contribute to this scale-up are: 1) dimensionality reduction through sampling, 2) the fact that Turbo-SMT operates on sparse data throughout its lifetime, and 3) that Turbo-SMT is *highly parallelizable*. Figure 1

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

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

The reason why we keep r fixed to 4 is because we conducted the same experiment on a machine with 4 cores (showing the results in the appendix) and for consistency purposes we chose the same r for both experiments. In Section 7.3 we show, however, that increasing r is improving the accuracy and if we use a machine with as many cores as r, then we may do so entirely in parallel, without significantly deviating from the timings that we report in this section.

When we compare Turbo-SMT against one of the two baselines, we make sure that we use the corresponding baseline as the core solver for Turbo-SMT. We observed a certain variation of solutions achieved by the baselines, depending on the number of iterations of the algorithm. Thus, we chose to experiment for three different limits for the number of iterations for the baselines (also enforcing the same limit in the core solver of Turbo-SMT). In particular, all the experiments are executed for 100, 1000, and 10000 iterations[7]. Note that this number indicates the *maximum* number of iterations (denoted by `iter`), however the baseline algorithm may terminate before reaching this number of iterations, depending on its (random) initialization. The termination criterion for both baselines was either meeting the maximum number of iterations `iter`, or the difference of the objective function between two consecutive iterations to be less than 10^{-8}. Finally, for every combination of rank, sampling factor, and number of iterations, we run Turbo-SMT and the corresponding baseline 5 times, in order to account for multiple local minima as well as variation in the execution times.

Figure 6 shows the run times for CMTF-OPT and CMTF-ALS. In particular, we show the average run time, as well as the maximum run time, in order to account for the worst case scenario. As expected, the run time for both baselines increases as the decomposition rank, and the number of iterations increase. There are a few cases where the runtime for `iter`=1000 is larger than that of `iter`=10000, however, since `iter` is merely the *maximum* number of iterations, and the problem being solved is a highly non-convex one, it is very likely that a run from the batch where `iter` was 10000 converged faster than a run from the batch of `iter`=1000. In Figure 7 we show the corresponding run times for Turbo-SMT, using both baselines. We observe the general trend of the run time increasing with the rank and the number of iterations, however, the run time decreases as the sampling factor decreases.

[7]10000 iterations is the default for CMTF-OPT, according to the documentation.

7.3 Accuracy

During the experiment described in Section 7.2, besides run time, we also measured the reconstruction error for each method and each combination of parameters. In Figure 8 we show the reconstruction errors for the two baselines, and in Figure 9 the error for Turbo-SMT. A first observation is that the error is fairly high for both the baselines and Turbo-SMT, fact that possibly indicates that the structure of the data is not exactly the one being imposed by CMTF, however, a low rank approximation is still useful for exploratory analysis.

Figures 9 assumes that the number of repetitions r is fixed to 4, which is a rather small number, considering the size of the data. When we, however, increase r, the approximation improves. In Figure 10 we demonstrate that the algorithm operates correctly, in the sense that it reduces the model error (Equation 2.1) when doing more repetitions. In particular, the vertical axis displays the relative error, i.e. $\frac{\text{Turbo}-\text{SMT error}}{\text{baseline error}}$ (with ideal being equal to 1) and the horizontal axis is the number of repetitions in the sampling. We use 60 × 200 × 9 portion of the BrainQ tensor and the entire matrix, to ensure that both the baseline (CMTF-OPT in this case) and Turbo-SMT run in a short amount of time. We set $s=[\ 10 \quad 20 \quad 1\ ]$. We run the experiment 1000 times and we keep the solutions achieving the minimum error. As Figure 10 shows, Turbo-SMT's relative error decreases as a function of the repetitions, indicating that Turbo-SMT generally reduces the CMTF approximation error with every repetition. As the decomposition rank increases, the number of repetitions needed increases as well, since the model to be captured becomes more complex. Because we are operating on smaller pieces of data, we aim for very low rank decompositions, in order to avoid overfactoring (i.e. choosing a higher number than the rank of the data) which may cause problems with the stitching and lead to instabilities. Additionally, due to the fact that Turbo-SMT operates on random samples of the data, convergence of the approximation error is noisier than deterministic methods such as ALS.

It is important to note that given a large dataset, during the first few repetitions of Turbo-SMT, the accuracy may be comparable to the null model's (i.e. the all-zero model), as shown in Figure 10; this is because Turbo-SMT starts with a null model and progressively builds it up. However, the important take home point here is that given a machine or a cluster of machines with enough amount of cores, we can enable the baselines to work on very large datasets that may not fit in main memory, and by running more repetitions, we explore more of the data and improve the approximation accuracy of the model.

7.4 Sparsity

One of the main advantages of Turbo-SMT is that, it is triple-sparse, i.e. starting from (possibly) sparse data, every intermediate result of Turbo-SMT is sparse, as well as the final output. In Fig. 11 we demonstrate the sparsity of Turbo-SMT's results by introducing the relative sparsity metric; this intuitive metric is simply the ratio of the output size of the baseline algorithm, divided by the output size of Turbo-SMT. The output size is simply calculated by adding up the number of non-zero entries for all factor matrices output by the algorithm. We use a portion of the BrainQ dataset in order to execute this experiment. We

can see that for the relatively dense BrainQ dataset, we obtained significantly more sparse results;, e.g. up to 65 times more sparse with almost same approximation error, for the case of CMTF-OPT. We observe a large difference of result sparsity when using CMTF-OPT, as opposed to ALS; most likely, this difference is due to the fact that, according to [4], CMTF-OPT converges to a better local minimum than ALS. The results of Fig. 11 indicate that our triple-sparse algorithm is able to capture the most useful variation of the data, successfully suppressing noise.

7.5 Comparison to Tucker3 Compression

One existing approach used for speeding up the PARAFAC decomposition is based on the CANDELINC theorem [16] and has been used in [15, 14]; roughly, the method first uses Tucker3 in order to compress the original tensor to a smaller, core tensor, then fits the PARAFAC decomposition on the core tensor, and finally, projects the factor matrices to the original space. In Section 5.3 of [31], it is implied that one could do the same for the CMTF problem. Before proceeding with a brief sketch of the approach which, to the best of our knowledge, *has not been published yet*, we provide a brief overview of the Tucker3 decomposition.

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 **U, V, Z** with sizes $I \times P$, $J \times Q$ and $K \times R$ respectively. The Tucker3 objective function is:

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

and we may write the decomposition, compactly, as:

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

Having a tensor $\underline{\mathbf{X}}$ coupled with matrices $\mathbf{Y}_i$, $i = 1 \cdots 3$, we may first obtain the Tucker3 decomposition of **X**. Consequently, we may use **U** in order to project $\mathbf{Y}_1$ to the compressed space, and respectively **V** for $\mathbf{Y}_2$ and **Z** for $\mathbf{Y}_3$. We, thus, obtain a new set of coupled data: the core tensor $\underline{\mathbf{G}}$, and the projected side matrices. Then, we fit a CMTF model to the compressed data, and as a final step, we use **U** in order to project **A, D** to their original dimension (and accordingly for the rest of the factor matrices).

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

- Tucker3 is now a bottleneck; its computation (even though there exist memory efficient implementations in the literature [10], [29], which we use in our implementation) is very costly, compared to the simple sampling scheme that Turbo-SMT is using. Even though alternatives to Tucker3 could be used for the compression (e.g. HOSVD [20]), the

authors of [15] state that the algorithm is better off using Tucker3 compression, quality-wise, whenever Tucker3 is able to be computed.

- This method, in contrast to Turbo-SMT, is not parallelizable, at least not in an obvious way, that would make its computation more efficient.
- This compression-based technique is not triple-sparse: The output of Tucker3 is dense, hence the core tensor $\underline{\mathbf{G}}$ and the projected side matrices are going to be dense. Additionally, both ALS and CTMF-OPT [4] produce dense factors. Therefore, this technique is prone to storage and interpretability issues.

We implemented this compression-based technique, using Tensor Toolbox's [10] memory efficient implementation of the Tucker3 decomposition. In Fig. 12, we illustrate the wall-clock time of this approach, compared to Turbo-SMT, on the entire BrainQ dataset; we chose $s = 5$ for Turbo-SMT, and we chose $P = Q = R = 60$ for the compression-based technique. We observe that Turbo-SMT performs significantly better, while, additionally, producing sparse outputs.

7.6 Performance of ALS with missing values

In order to measure resilience to missing values we define the *Signal-to-Noise Ratio* (SNR) as simply as $\mathrm{SNR} = \frac{\|\underline{\mathbf{X}}_m\|_F^2}{\|\underline{\mathbf{X}}_m - \underline{\mathbf{X}}_0\|_F^2}$, where $\underline{\mathbf{X}}_m$ is the reconstructed tensor when a m fraction of the values are missing. In Figure 13, we demonstrate the results of that experiment; we observe that even for a fair amount of missing data, the algorithm performs reasonably well, achieving high SNR. Moreover, for small amounts of missing data, the speed of the algorithm is not degraded, while for larger values, it is considerably slower, probably due to Matlab's implementation issues. However, this is encouraging, in the sense that if the amount of missing data is not overwhelming, Turbo-SMT is able to deliver a very good approximation of the latent subspace. This experiment was, again, conducted on a portion of BrainQ.

8 Related Work

8.1 Coupled, Multi-block, Multi-set Models

Coupled Matrix-Tensor Factorizations belong to a family of models also referred to as *Multi-block* or *Multi-set* in the literature. Smilde et al. in [48] provided the first disciplined treatment of such multi-block models, in a chemometrics context. One of the earliest works that introduce the concept of coupling in data mining applications is [11], where the authors apply their algorithms to movie recommendation and newsgroup article clustering. In [47], Singh and Gordon introduce a collective matrix factorization framework, again in a data mining setting, where the coupling is between matrices. An important issue with these models is how to weigh the different data blocks such that scaling differences may be alleviated. In [52], Wilderjans et al. propose and compare two different weighing schemes.

Most related to the present work is the work of Acar et al. in [4], where a first order optimization approach is proposed, in order to solve the CMTF problem. In [5], Acar et. al apply the CMTF model, using the aforementioned first-order approach in a bioinformatics setting. In [3], Acar et. al introduce a coupled matrix decomposition, where two matrices match on one of the two dimensions, and are decomposed in the same spirit as in CMTF, while imposing explicit sparsity constraints (via ℓ_1 norm penalties); although Turbo-SMT also produces sparse factors, this so happens as a fortuitous byproduct of sampling, whereas in [3] an explicit sparsity penalty is considered. As an interesting application, in [55], the authors employ CMTF for Collaborative Filtering. On a related note, [53], [33], and [35] introduce models where multiple tensors are coupled with respect to one mode, and analyzed jointly. Finally, extending the CMTF framework, Acar et al. propose in [7] a new model where rank-one components can be shared by the matrix and the tensor, or belong solely to either one of the data pieces/blocks.

To the best of our knowledge, Turbo-SMT is the first algorithm that is able parallelize and sparsify any (possibly highly fine tuned) core algorithm for CMTF. The original paper introducing Turbo-SMT appeared at SIAM SDM 2014 [45].

8.2 Fast & Scalable Tensor Decompositions

The mechanics behind the Tensor Toolbox for Matlab [10] are introduced in [30] and later on in[9], where Bader and Kolda first show how to efficiently decompose sparse tensors, avoiding the materialization of prohibitively dense and large intermediate data. Bro and Sidiropoulos introduce a compression based method in [14] which is able to speed up PARAFAC, without however exploiting parallelization. In [41] the authors introduced a parallel algorithm for the regular PARAFAC decomposition, where a sampling scheme of similar nature as here is exploited; following a similar split and merge parallelization scheme, [46] introduces a fast and parallelizable algorithm for PARAFAC, based on compression using random projections, which provides guarantees with respect to the identifiability of the true latent factors. In [26], a scalable MapReduce implementation of PARAFAC is presented, where the intermediate data problem is avoided in the same way as in [30, 9], with specific considerations that pertain to the distributed implementation. Finally, in [54], the authors introduce a parallel framework in order to handle tensor decompositions efficiently. More recently, Beutel et al. [12] introduce a highly scalable Distributed Stochastic Gradient system for CMTF on MapReduce. Finally, in [19], the authors propose an alternative parallel architecture for decomposing tensors on a cluster, by exploiting the multilinear nature of the tensor data and mapping the computation in a similar multilinear fashion on the cluster's machines. Finally, in [28] the authors introduce a method to vertically partition a multi-relational dataset (which can be viewed as a multiway tensor) using relational normalization, which results in independent decompositions of smaller tensors, that are combined at the end.

8.3 Sampling based techniques

Turbo-SMT draws its strength in reducing the problem size using biased sampling. In that sense, it is related to the line of work under the name of "CUR" [22]. In its matrix version, given a matrix $\mathbf{X}$, CUR samples m columns of $\mathbf{X}$ which define matrix $\mathbf{C}$, samples n rows of

X that form matrix **R** and uses **C, R** to compute the "core" matrix **U**. The main advantage of doing so is that **C, R** contain actual pieces of the data, which makes the decomposition easily interpretable. There is also follow up work that generalizes CUR to tensors [36, 23] in the same spirit. The way sampling is used in Turbo-SMT is different, since it is used in order to obtain a small piece of the data which is later decomposed according to CMTF; thus the end result of the algorithm will be approximating the result of CMTF and not define a decomposition that contains actual pieces of the data.

Finally, in [41] a subset of the authors of the present work propose a biased sampling based PARAFAC decomposition that uses the same sample, parallelize, and merge framework.

8.4 Tensor Applications to Brain Data

There has been substantial related work, which utilizes tensors for analyzing brain data. In [2] Acar et al. apply the PARAFAC decomposition on EEG measurements in order to detect epileptic seizures. Morup et al. in [38] introduce a variation of the PARAFAC decomposition that is able to handle temporal shifts across brain measurements. In [17], the authors describe how one can use non-negative tensor factorization for source separation and analysis of brain signals, and finally, in [18] et al. introduce a constrained PARAFAC model in order to identify the connectivity between different brain regions.

8.5 Tensor Applications to Social Network Analysis

Social network analysis has also been tackled using tensors before. One of the first works that did so was the work of Bader et al. [8], where the authors detect cliques of dense interaction in the ENRON e-mail network, while tracking their activity over time. The findings of [8] have also been reproduced in [43] using different, albeit in the tensor regime, techniques. Lin et al. [33] employs coupled tensor factorization for community detection in social graphs with context information about the users. Finaly, in [41], the authors apply their tensor decomposition algorithm on a Facebook dataset, detecting normal and abnormal user interaction.

9 Conclusions

The main contributions of our work are:

- *Parallel & triple-sparse algorithm:* Turbo-SMT is able to parallelize *any* CMTF solver, producing much sparser results. Moreover, the way that Turbo-SMT is defined, it can operate on data that do not fit in main memory, thus enabling state of the art CMTF solvers to work on such data, even though they are not specifically designed to do so.
- *Effectiveness and Knowledge Discovery*: Turbo-SMT, applied to the BrainQ dataset, discovers meaningful triple-mode clusters: clusters of words, of questions, and of brain regions have similar behavior; as a by-product, Turbo-SMT predicts brain activity, with performance that matches state of the art predictors.

- *Reproducibility*: We make our code public, enabling reproducibility and re-usability of our work.

Acknowledgments

Research was funded by grants NSF IIS-1247489, NSF IIS-1247632, NSF CDI 0835797, NIH/NICHD 12165321, DARPA FA87501320005, and a gift from Google. 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. The authors would also like to thank Leila Wehbe and Alona Fyshe for their initial help with the BrainQ data. Finally, the authors would like to thank Dr. Tamara Kolda whose useful feedback led to the comparison of Turbo-SMT against the Tucker3 compression based algorithm.

References

1. Read the web. http://rtw.ml.cmu.edu/rtw/.
2. Acar E, Aykut-Bingol C, Bingol H, Bro R, Yener B. Multiway analysis of epilepsy tensors. Bioinformatics. 2007; 23(13):i10–i18. [PubMed: 17646285]
3. Acar, E.; Gurdeniz, G.; Rasmussen, MA.; Rago, D.; Dragsted, LO.; Bro, R. IEEE ICDM Workshops. IEEE; 2012. Coupled matrix factorization with sparse factors to identify potential biomarkers in metabolomics; p. 1-8.
4. Acar E, Kolda TG, Dunlavy DM. All-at-once optimization for coupled matrix and tensor factorizations. 2011 arXiv preprint arXiv:1105.3422.
5. Acar E, Plopper GE, Yener B. Coupled analysis of in vitro and histology tissue samples to quantify structure-function relationship. PloS one. 2012; 7(3):e32227. [PubMed: 22479315]
6. Acar, Evrim; Dunlavy, Daniel M.; Kolda, Tamara G.; Mørup, Morten. Scalable tensor factorizations for incomplete data. Chemometrics and Intelligent Laboratory Systems. 2011; 106(1):41–56.
7. Acar, Evrim; Arendt Rasmussen, Morten; Savorani, Francesco; Næs, Tormod; Bro, Rasmus. Understanding data fusion within the framework of coupled matrix and tensor factorizations. Chemometrics and Intelligent Laboratory Systems. 2013; 129:53–63.
8. Bader, Brett W.; Harshman, Richard A.; Kolda, Tamara G. Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on. IEEE; 2007. Temporal analysis of semantic graphs using asalsan; p. 33-42.
9. Bader, Brett W.; Kolda, Tamara G. Efficient matlab computations with sparse and factored tensors. SIAM Journal on Scientific Computing. 2007; 30(1):205–231.
10. Bader, BW.; Kolda, TG. Matlab tensor toolbox version 2.2. Albuquerque, NM, USA: Sandia National Laboratories; 2007.
11. Banerjee, Arindam; Basu, Sugato; Merugu, Srujana. SDM. Vol. 7. SIAM; 2007. Multi-way clustering on relation graphs; p. 145-156.
12. Beutel, Alex; Kumar, Abhimanu; Papalexakis, Evangelos; Talukdar, Partha Pratim; Faloutsos, Christos; Xing, Eric P. Flexifact: Scalable flexible factorization of coupled tensors on hadoop. 2014
13. Bro, R. PhD thesis. Københavns Universitet; 1998. Multi-way analysis in the food industry: models, algorithms, and applications.
14. Bro, R.; Sidiropoulos, ND.; Giannakis, GB. Int. Workshop Independent Component and Blind Signal Separation Anal. 1999. A fast least squares algorithm for separating trilinear mixtures; p. 11-15.
15. Bro, Rasmus; Andersson, Claus A. Improving the speed of multiway algorithms: Part ii: Compression. Chemometrics and Intelligent Laboratory Systems. 1998; 42(1):105–113.
16. Douglas Carroll J, Pruzansky Sandra, Kruskal Joseph B. Candelinc: A general approach to multidimensional analysis of many-way arrays with linear constraints on parameters. Psychometrika. 1980; 45(1):3–24.
17. Cichocki, Andrzej; Zdunek, Rafal; Huy Phan, Anh; Amari, Shun-ichi. Nonnegative matrix and tensor factorizations: applications to exploratory multi-way data analysis and blind source separation. Wiley; 2009.

18. Davidson, Ian; Gilpin, Sean; Carmichael, Owen; Walker, Peter. Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM; 2013. Network discovery via constrained tensor analysis of fmri data; p. 194-202.
19. De Almeida, André LF.; Kibangou, Alain Y., et al. Distributed large-scale tensor decomposition; Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on; 2014.
20. De Lathauwer, Lieven; De Moor, Bart; Vandewalle, Joos. A multilinear singular value decomposition. SIAM journal on Matrix Analysis and Applications. 2000; 21(4):1253–1278.
21. Deerwester S, Dumais ST, Furnas GW, Landauer TK, Harshman R. Indexing by latent semantic analysis. Journal of the American Society for Information Science. 1990 Sep; 41(6):391–407.
22. Drineas P, Kannan R, Mahoney MW, et al. Fast monte carlo algorithms for matrices iii: Computing a compressed approximate matrix decomposition. SIAM Journal on Computing. 2006; 36(1):184.
23. Drineas, Petros; Mahoney, Michael W. A randomized algorithm for a tensor-based generalization of the singular value decomposition. Linear algebra and its applications. 2007; 420(2):553–571.
24. Harshman RA. Foundations of the parafac procedure: Models and conditions for an ”explanatory” multimodal factor analysis. 1970
25. Hung C, Markham TL. The moore-penrose inverse of a partitioned matrix m= adbc. Linear Algebra and its Applications. 1975; 11(1):73–86.
26. Kang, U.; Papalexakis, E.; Harpale, A.; Faloutsos, C. SIGKDD. ACM; 2012. Gigatensor: scaling tensor analysis up by 100 times-algorithms and discoveries; p. 316-324.
27. Kiers HAL. Towards a standardized notation and terminology in multiway analysis. Journal of Chemometrics. 2000; 14(3):105–122.
28. Kim, Mijung; Candan, K Selçuk. Proceedings of the 21st ACM international conference on Information and knowledge management. ACM; 2012. Decomposition-by-normalization (dbn): leveraging approximate functional dependencies for efficient tensor decomposition; p. 355-364.
29. Kolda, Tamara G.; Sun, Jimeng. Data Mining, 2008. ICDM’08. Eighth IEEE International Conference on. IEEE; 2008. Scalable tensor decompositions for multi-aspect data mining; p. 363-372.
30. Kolda TG, Bader BW. The tophits model for higher-order web link analysis. Workshop on Link Analysis, Counterterrorism and Security. 2006; 7:26–29.
31. Kolda TG, Bader BW. Tensor decompositions and applications. SIAM review. 2009; 51(3)
32. Kuhn, Harold W. The hungarian method for the assignment problem. Naval research logistics quarterly. 1955; 2(1–2):83–97.
33. Lin, YR.; Sun, J.; Castro, P.; Konuru, R.; Sundaram, H.; Kelliher, A. SIGKDD. ACM; 2009. Metafac: community discovery via relational hypergraph factorization; p. 527-536.
34. Liu S, Trenkler G. Hadamard, khatri-rao, kronecker and other matrix products. International Journal of Information and Systems Sciences. 2008:160–177.
35. Liu, W.; Chan, J.; Bailey, J.; Leckie, C.; Ramamohanarao, K. SDM 2013. SIAM; 2013. Mining labelled tensors by discovering both their common and discriminative subspaces.
36. Mahoney, MW.; Maggioni, M.; Drineas, P. SIGKDD. ACM; 2006. Tensor-cur decompositions for tensor-based data; p. 327-336.
37. Mitchell TM, Shinkareva SV, Carlson A, Chang KM, Malave VL, Mason RA, Just MA. Predicting human brain activity associated with the meanings of nouns. Science. 2008; 320(5880):1191–1195. [PubMed: 18511683]
38. Mørup, Morten; Kai Hansen, Lars; Arnfred, Sidse Marie; Lim, Lek-Heng; Hougaard Madsen, Kristoffer. Shift-invariant multilinear decomposition of neuroimaging data. NeuroImage. 2008; 42(4):1439–1450. [PubMed: 18625324]
39. Brian, Murphy; Talukdar, Partha; Mitchell, Tom. Selecting corpus-semantic models for neurolinguistic decoding; First Joint Conference on Lexical and Computational Semantics (* SEM); 2012. p. 114-123.
40. Palatucci, Mark; Pomerleau, Dean; Hinton, Geoffrey; Mitchell, Tom. Zero-shot learning with semantic output codes. Advances in neural information processing systems. 2009; 22:1410–1418.

41. Papalexakis E, Faloutsos C, Sidiropoulos N. Parcube: Sparse parallelizable tensor decompositions. Machine Learning and Knowledge Discovery in Databases. 2012:521–536.
42. Papalexakis, EE.; Sidiropoulos, ND. Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on. IEEE; 2011. Co-clustering as multilinear decomposition with sparse latent factors; p. 2064-2067.
43. Papalexakis EE, Sidiropoulos ND, Bro R. From k- means to higher-way co-clustering: Multilinear decomposition with sparse latent factors. Signal Processing, IEEE Transactions on. 2013; 61(2): 493–506.
44. Papalexakis, Evangelos E.; Mitchell, Tom M.; Sidiropoulos, Nicholas D.; Faloutsos, Christos; Talukdar, Partha Pratim; Murphy, Brian. Scoup-smt: Scalable coupled sparse matrix-tensor factorization. 2013 arXiv preprint arXiv: 1302.7043.
45. Papalexakis, Evangelos E.; Mitchell, Tom M.; Sidiropoulos, Nicholas D.; Faloutsos, Christos; Talukdar, Partha Pratim; Murphy, Brian. Turbo-smt: Accelerating coupled sparse matrix-tensor factorizations by 200x. SIAM SDM. 2014
46. Sidiropoulos, ND.; Papalexakis, EE.; Faloutsos, C. A parallel algorithm for big tensor decomposition using randomly compressed cubes (paracomp); Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on; 2014.
47. Singh, Ajit P.; Gordon, Geoffrey J. Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM; 2008. Relational learning via collective matrix factorization; p. 650-658.
48. Smilde AK, Westerhuis JA, Boque R. Multiway multiblock component and covariates regression models. Journal of Chemometrics. 2000; 14(3):301–331.
49. Tomasi G, Bro R. Parafac and missing values. Chemometrics and Intelligent Laboratory Systems. 2005; 75(2):163–180.
50. Tomasi, Giorgio; Bro, Rasmus. A comparison of algorithms for fitting the parafac model. Computational Statistics & Data Analysis. 2006; 50(7):1700–1734.
51. Viswanath, Bimal; Mislove, Alan; Cha, Meeyoung; Gummadi, Krishna P. On the evolution of user interaction in facebook. SIGCOMM Workshop on Social Networks. 2009
52. Wilderjans T, Ceulemans E, Van Mechelen I. Simultaneous analysis of coupled data blocks differing in size: A comparison of two weighting schemes. Computational Statistics & Data Analysis. 2009; 53(4):1086–1098.
53. Yokota, Tatsuya; Cichocki, Andrzej; Yamashita, Yukihiko. Neural Information Processing. Springer; 2012. Linked parafac/cp tensor decomposition and its fast implementation for multi-block tensor analysis; p. 84-91.
54. Zhang Q, Berry M, Lamb B, Samuel T. A parallel nonnegative tensor factorization algorithm for mining global climate data. Computational Science–ICCS 2009. 2009:405–415.
55. Zheng, Vincent W.; Cao, Bin; Zheng, Yu; Xie, Xing; Yang, Qiang. Collaborative filtering meets mobile recommendation: A user-centered approach. AAAI. 2010

Appendix

A Run times of BrainQ on a less powerful machine

The run time experiments shown in Section 7.2 are executed on a powerful workstation with ample memory and processing power. In this appendix, we show results of the very same experiments, when executed on a much smaller, more accessible machine, an iMac with 4Gb of RAM and an Intel i5 processor. The Matlab version on that machine was 7.12.0.635 (R2011a) 64-bit. Due to timing considerations, for the ALS algorithm we only ran the experiment for 100 iterations of the core solver, since the machines memory and processing power seemed inadequate.

Figure 14 shows the run times of the baselines. As a general trend, we observe that for most of the configurations, the algorithms take more time to execute compared to the powerful workstation case of Section 7.2, which is expected. In Figure 15 we show the timings for Turbo-SMT using the two baselines as core solvers; Turbo-SMT is generally faster than the baselines, however the speedup yielded here is smaller. Again, this behavior is expected, since the all four cores of the processor were fully utilized during the entire time of the experiment, and the memory of the machine was rather small, leading to a large swap file which resulted in heavy disk I/O, further burdening the CPU cores.

However, the main principle of Turbo-SMT still holds: by breaking up the data into smaller pieces, Turbo-SMT is able to parallelize the computation, improving performance and potentially enabling the analysis of very large datasets that do not fit in main memory that the state of the art simply cannot handle.

B Run times and sparsity of Facebook

In this section of the appendix, we provide more details on the technical aspects of analyzing the Facebook dataset, that do not pertain to data analysis and discovery. We were not able to measure reconstruction error in this dataset due to Matlab's memory limitations. Figure 16(a) shows the run time for CMTF-OPT and Turbo-SMT using CMTF-OPT for Facebook, for a variety of ranks (using $s_1 = 10$, $s_2 = 10$, and $s_3 = 2$, and 100 iterations for the core solver). Timing was computed on the same workstation that was used for Section 7.2. Because the Facebook dataset is extremely sparse and the baseline is able to handle it more efficiently, using sparse computations, the speedup observed is very small. However, as shown in Figure 16(b), Turbo-SMT yields results that are much sparser than those of the baseline (where the definition of sparsity is the same as in Section 7.4), which is very important for interpreting the results.

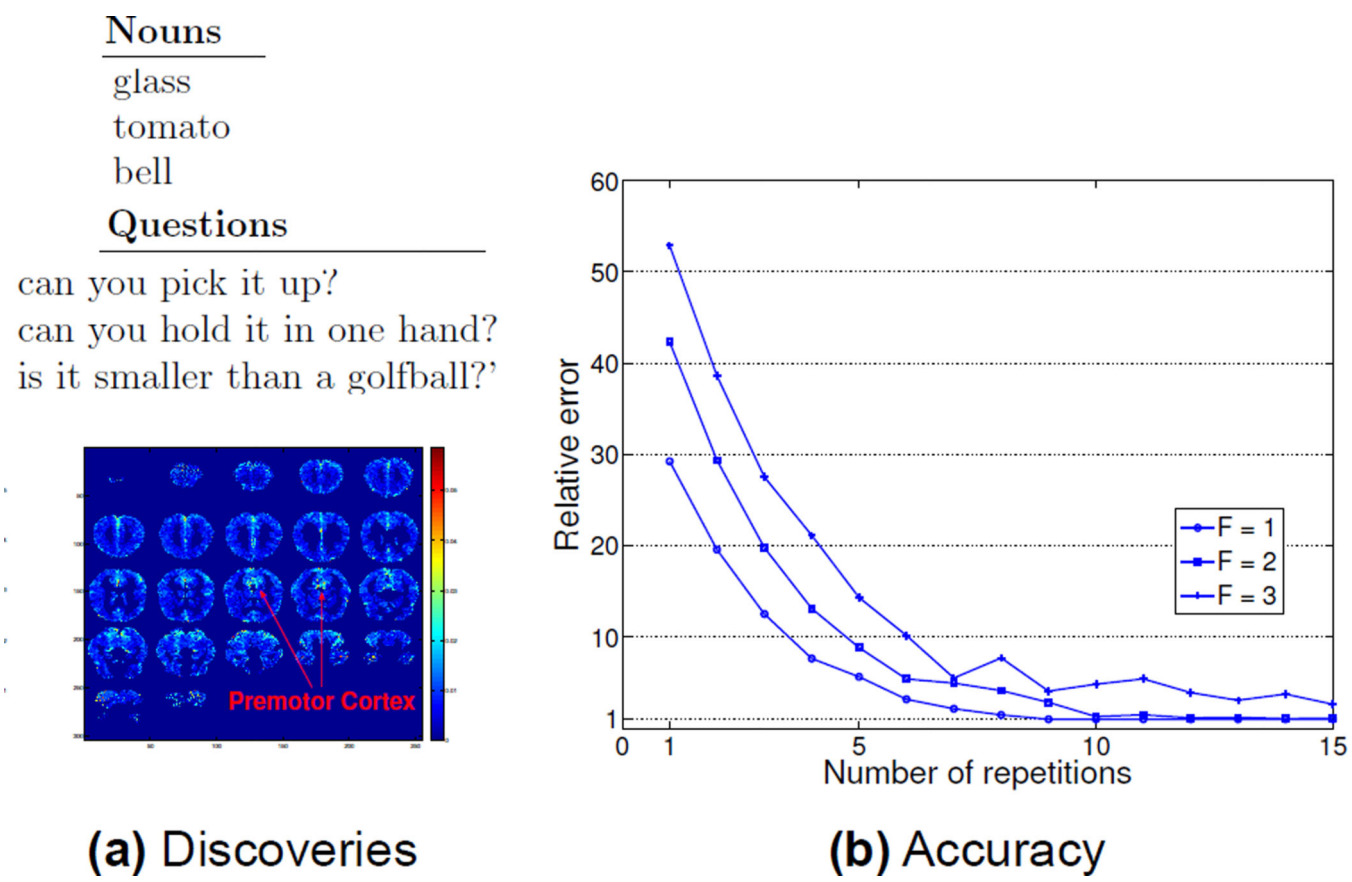


Figure 1.

(a): Analyzing the BrainQ dataset for the Neurosemantics application, finding semantically similar nouns that activate the same part of the brain. In this case, the premotor cortex is associated with movements such as holding or picking small items up. (b): The proposed algorithm Turbo-SMT reduces the approximation error, while running in parallel, and operating on much smaller pieces of data at any given point in time.

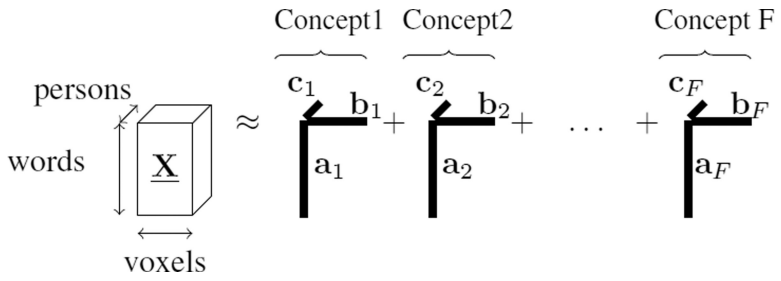


Figure 2.

PARAFAC decomposition [24] of a three-way tensor of a brain activity tensor as sum of F outer products (rank-one tensors), reminiscent of the rank-F singular value decomposition of a matrix. Each component corresponds to a **latent** concept of, e.g. “insects”, “tools” and so on, a set of brain regions that are most active for that particular set of words, as well as groups of persons.

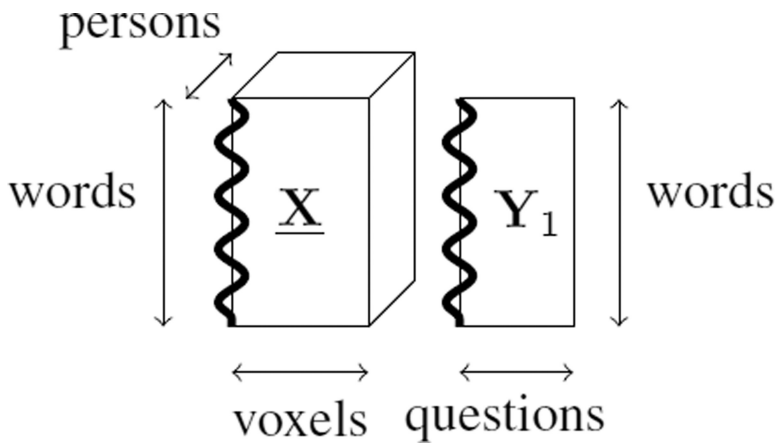


Figure 3.

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

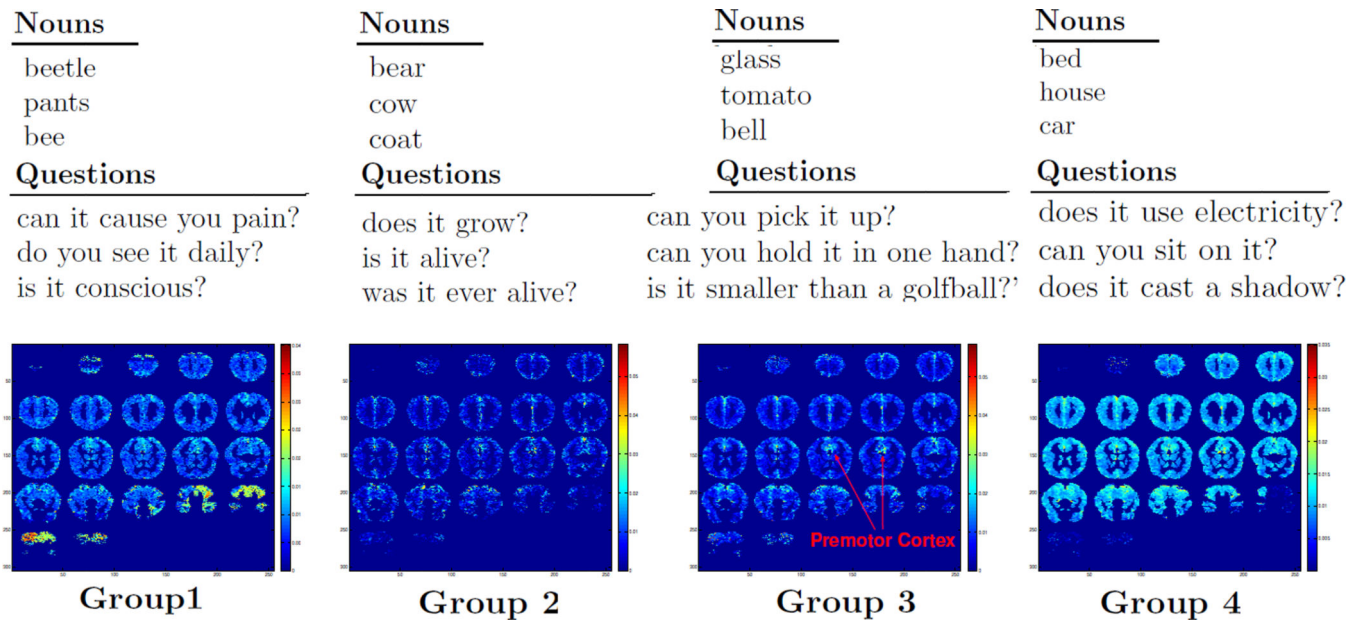


Figure 4.

Turbo-SMT finds meaningful groups of words, questions, and brain regions that are (both negatively and positively) correlated, as obtained using Turbo-SMT. For instance, Group 3 refers to small items that can be held in one hand, such as a tomato or a glass, and the activation pattern is very different from the one of Group 1, which mostly refers to insects, such as bee or beetle. Additionally, Group 3 shows high activation in the *premotor cortex* which is associated with the concepts of that group.

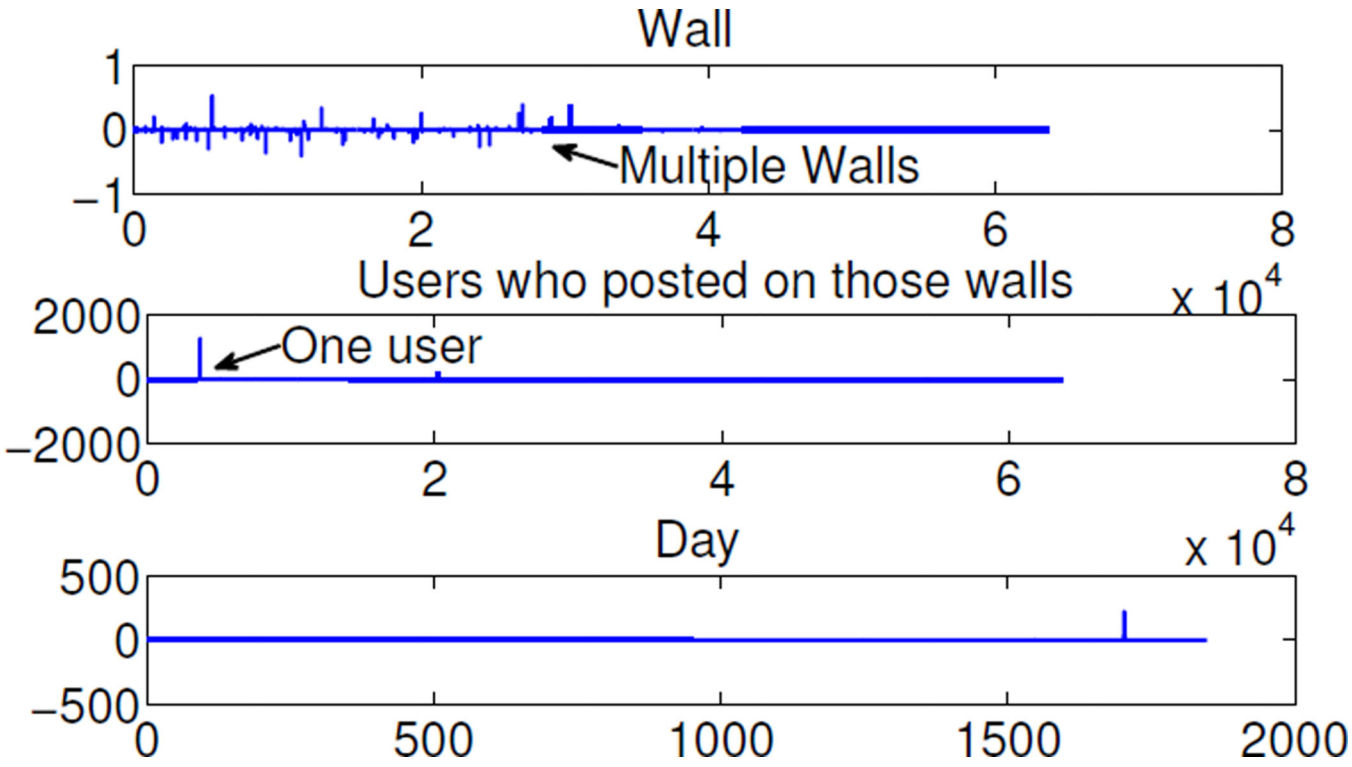
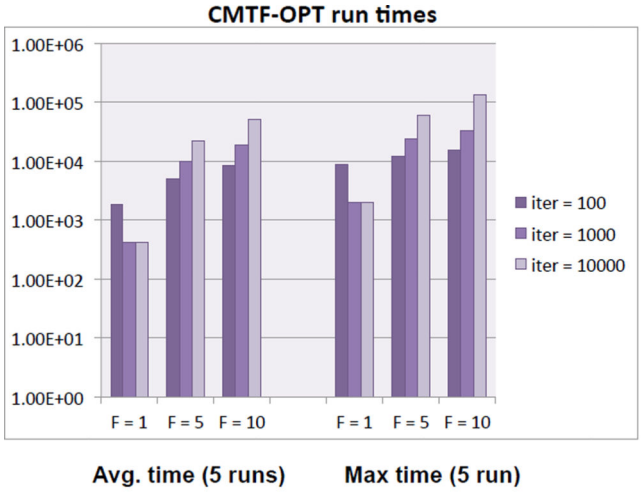
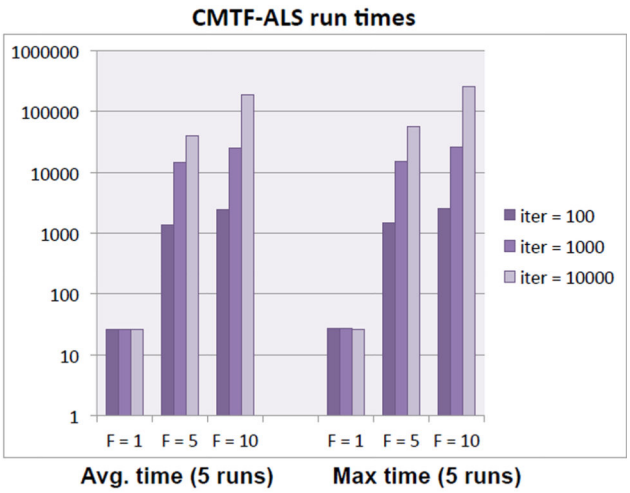


Figure 5.

This is a pattern extracted using Turbo-SMT, which shows what appears to be a spammer on the Facebook dataset: One person, posting to many different walls on a single day.



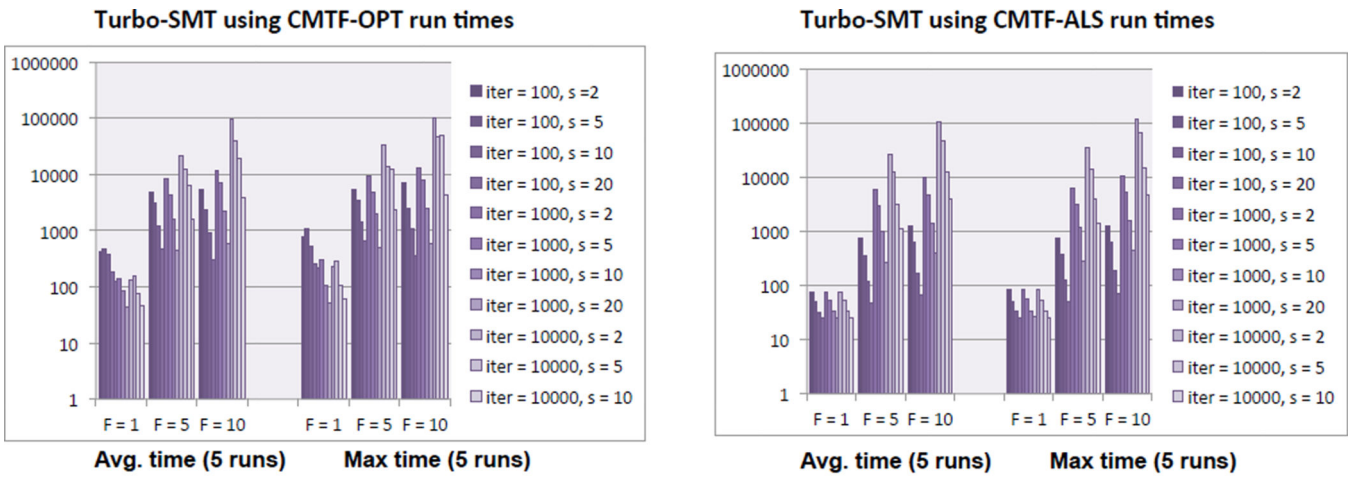
(a) CMTF_OPT



(b) CMTF_ALS

Figure 6.

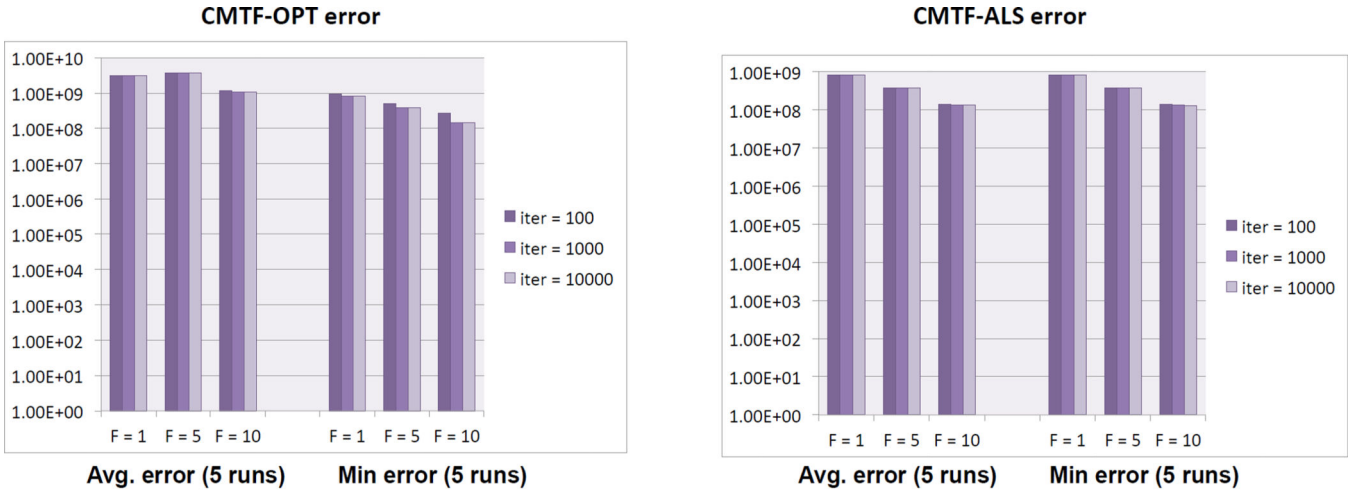
Run time (in seconds) for the baselines, on the full BrainQ dataset. The results shown are over 5 full runs of the algorithm.



(a) TURBO-SMT using CMTF_OPT (b) TURBO-SMT using CMTF_ALS

Figure 7.

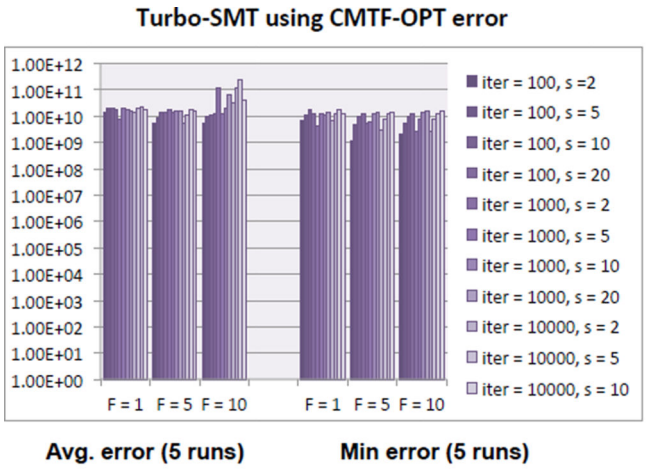
Run time (in seconds) for Turbo-SMT when using each of the baselines as core solver. The results shown are over 5 full runs of the algorithm. For every value of F, the bars on the plot starting from left to right, correspond to the parameters shown on the legend. The number of repetitions r was set to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



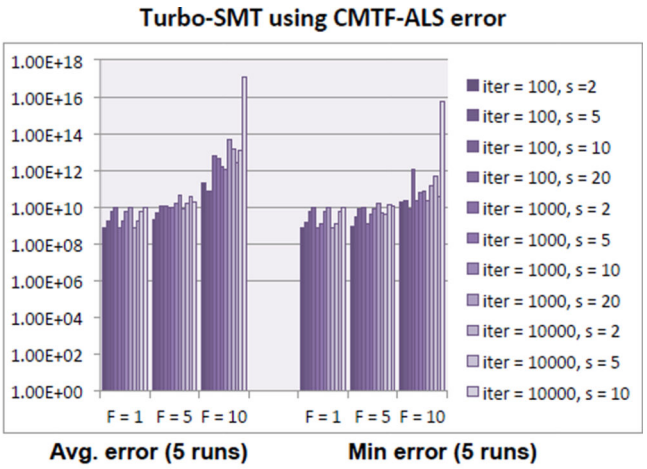
(a) CMTF_OPT (b) CMTF_ALS

Figure 8.

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

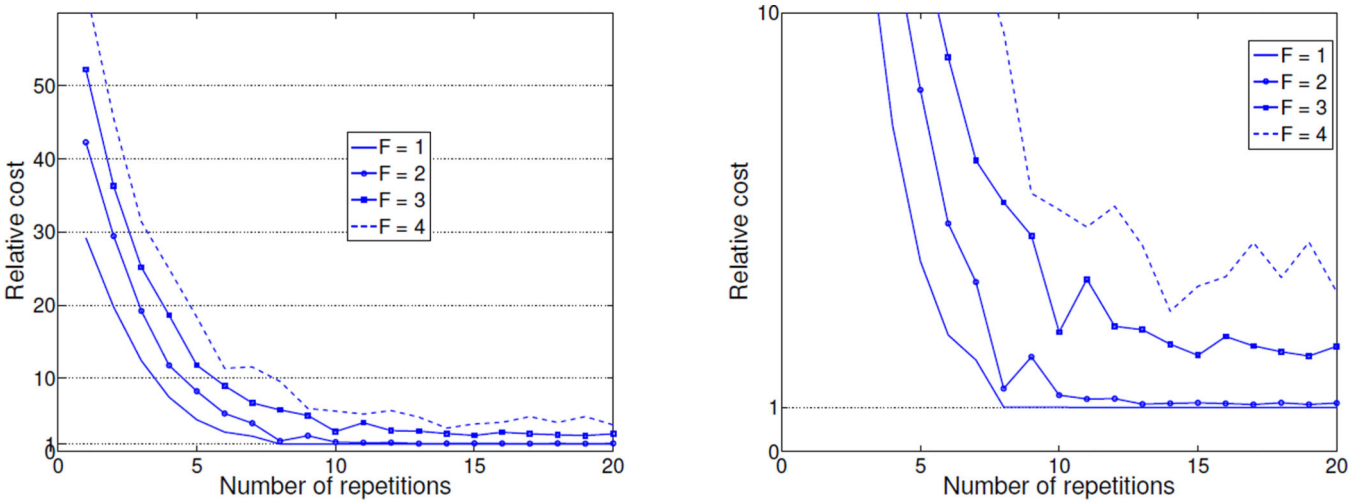


(a) TURBO-SMT using CMTF_OPT



(b) TURBO-SMT using CMTF_ALS

Figure 9.
Reconstruction error for Turbo-SMT.



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

Figure 10.

The relative error of the model, as a function of the number of repetitions r is decreasing, which empirically shows that Turbo-SMT generally reduces the approximation error of the CMTF model.

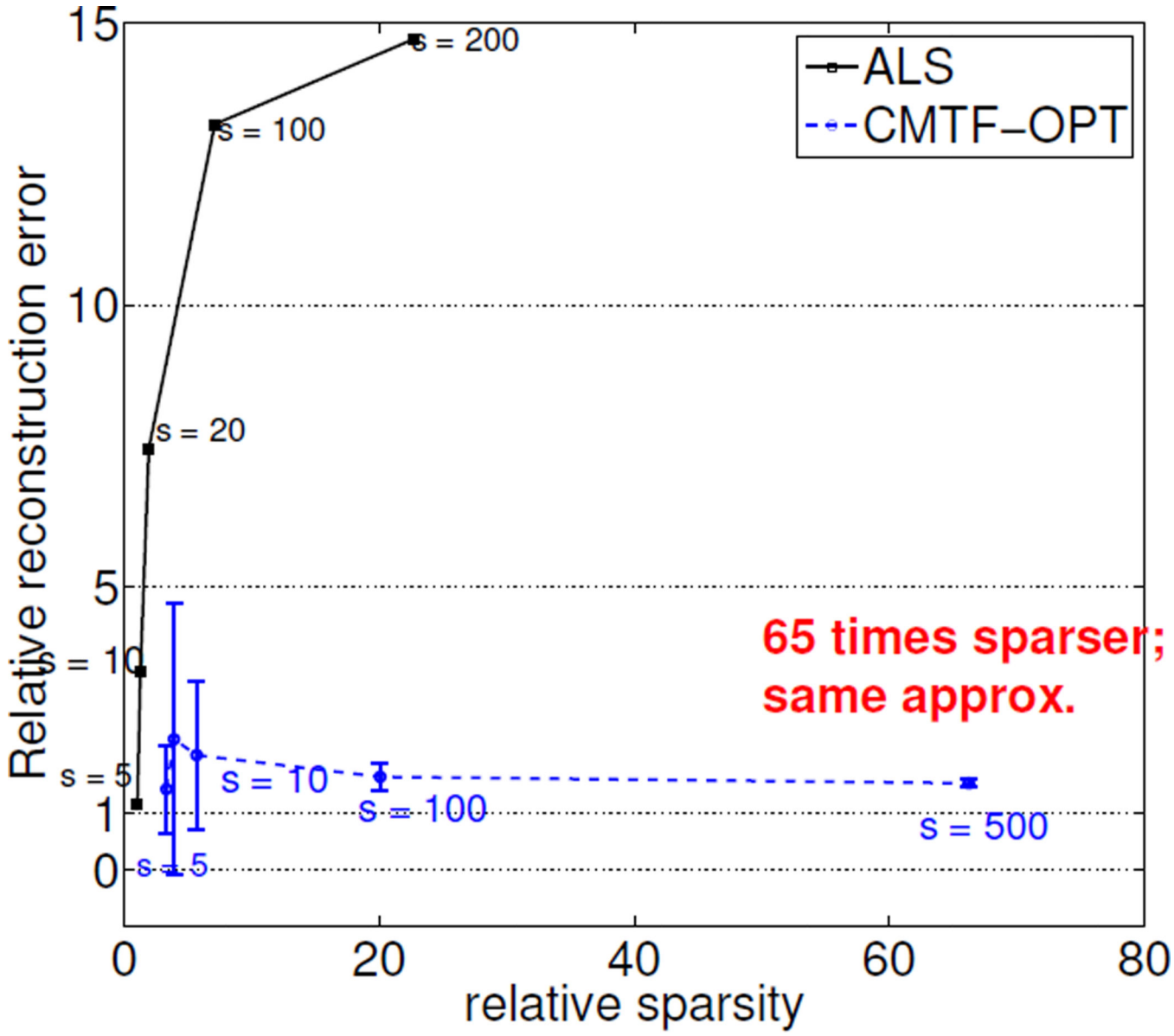


Figure 11.

The relative output size vs. the relative cost indicates that, even for very dense datasets such as BrainQ, we are able to get up to 65 fold (for CMTF-OPT) decrease in the output size, while maintaining almost same approximation error as the baseline.

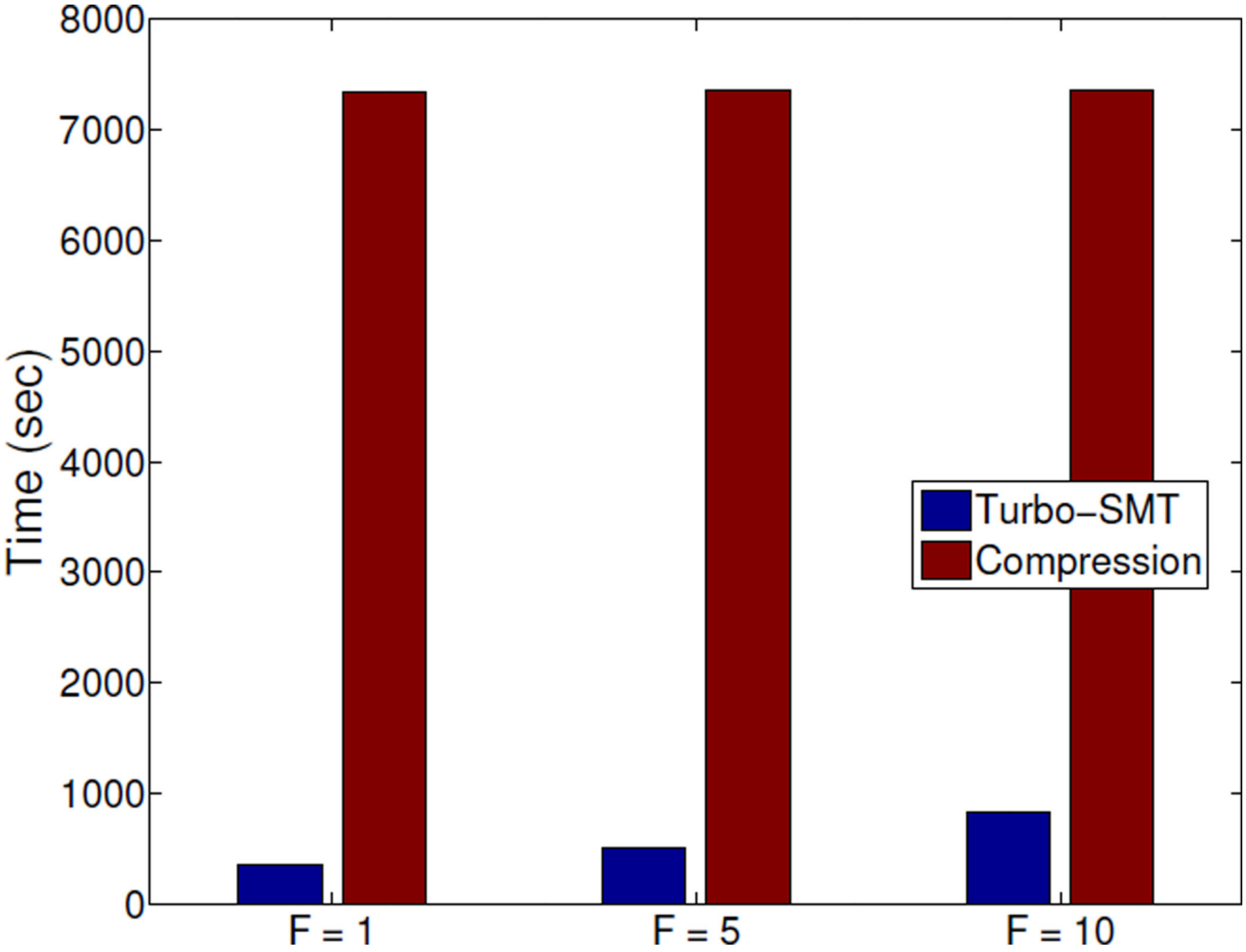


Figure 12.
Comparison of Turbo-SMT and a compression-based technique, that uses the Tucker3 decomposition, as described in Sec. 7.5. Turbo-SMT significantly outperforms the alternative method.

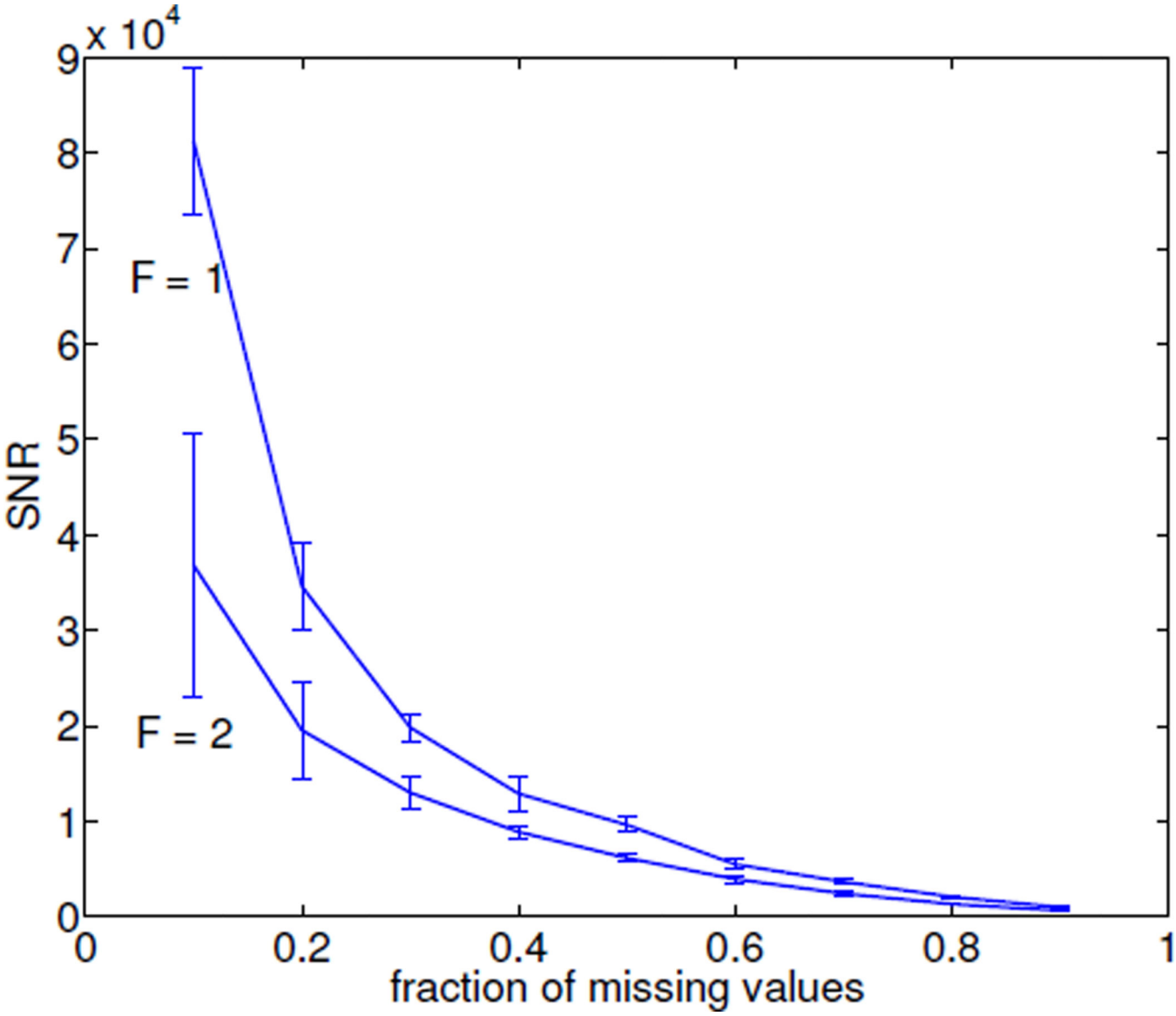
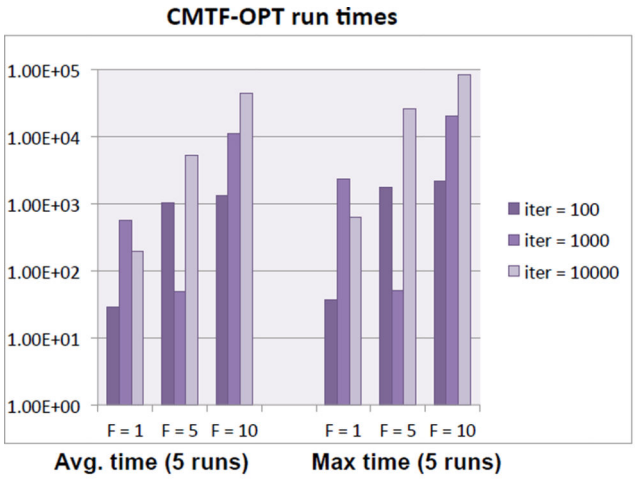
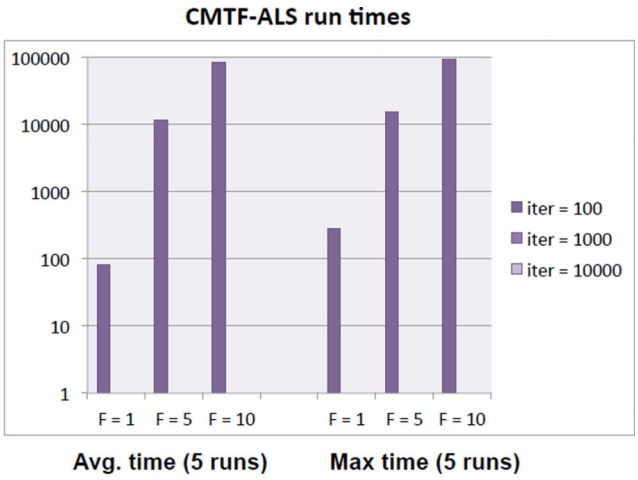


Figure 13.

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

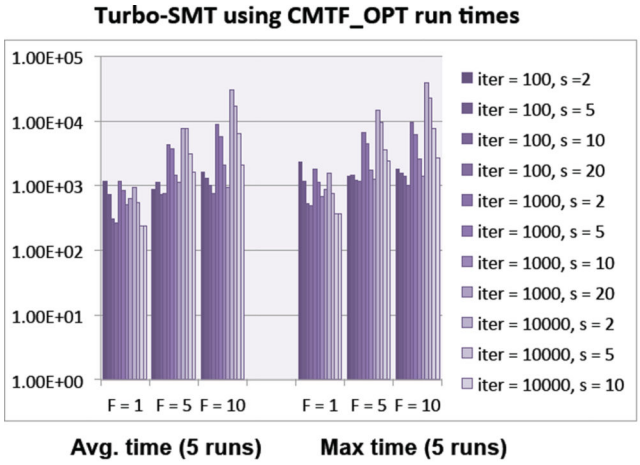


(a) CMTF_OPT

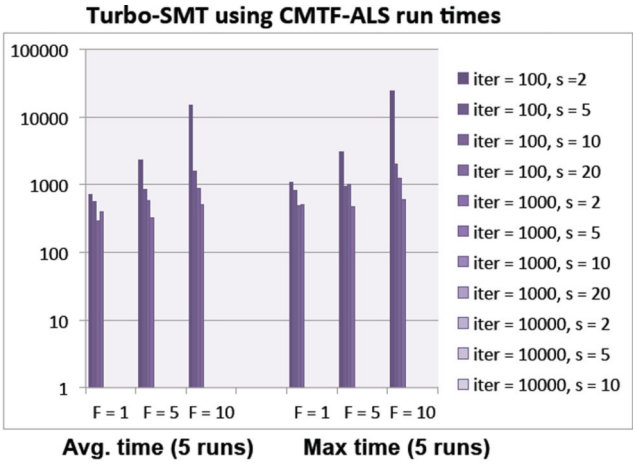


(b) CMTF_ALS

Figure 14.
Run time (in seconds) for the baselines

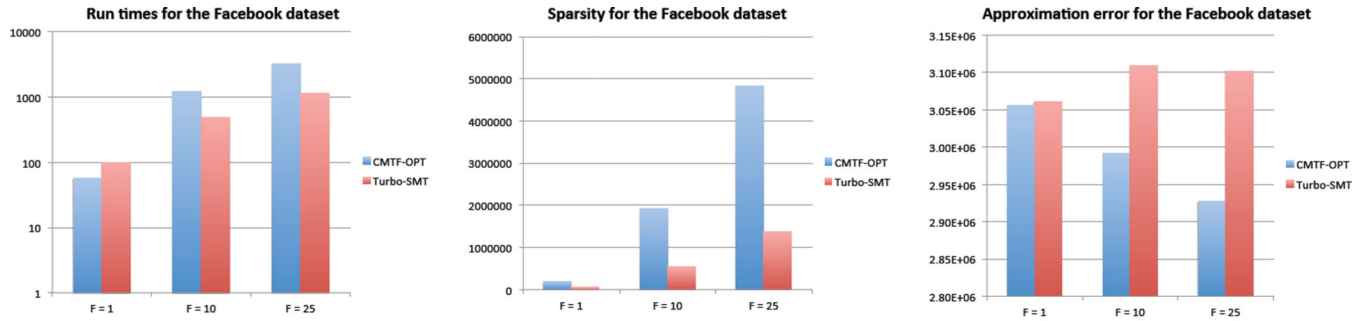


(a) CMTF_OPT



(b) CMTF_ALS

Figure 15.
Run time (in seconds) for Turbo-SMT



(a) Run time (in seconds) (b) Sparsity (c) Approximation error

Figure 16.

Run times, sparsity, and approximation error for the Facebook dataset, using CMTF-OPT and Turbo-SMT with CMTF-OPT as a core solver.

Table 1

Table of symbols

Symbol	Description
CMTF	Coupled Matrix-Tensor Factorization
ALS	Alternating Least Squares
CMTF-OPT	Algorithm introduced in [4]
x, $\mathbf{x}_:$, $\mathbf{X}$, $\underline{\mathbf{X}}$	scalar, vector, matrix, tensor (respectively)
$\mathbf{A} \odot \mathbf{B}$	Khatri-rao product.
$\mathbf{A} \otimes \mathbf{B}$	Kronecker product.
$\mathbf{A} * \mathbf{B}$	Hadamard (elementwise) product.
$\mathbf{A}^\dagger$	Pseudoinverse of **A** (see Sec. 2)
$\|\mathbf{A}\|_F$	Frobenius norm of **A**.
$\mathbf{a} \circ \mathbf{b} \circ \mathbf{c}$	$(\mathbf{a} \circ \mathbf{b} \circ \mathbf{c})(i, j, k) = \mathbf{a}(i)\mathbf{b}(j)\mathbf{c}(k)$
(i) as superscript	Indicates the i-th iteration
$\mathbf{A}_1^i, \mathbf{a}_1^i$	series of matrices or vectors, indexed by i.
$\underline{\mathbf{X}}_{(i)}$	i-th mode unfolding of tensor $\underline{\mathbf{X}}$ (see [27]).
$\mathcal{Q}$	Set of indices.
$\mathbf{x}(\mathcal{Q})$	Spanning indices $\mathcal{Q}$ of **x**.
nucleus	a biased random sample of the data tensor.

Table 2

Pseudoinversion speedup (100000 runs)

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