Turbo-SMT: Accelerating Coupled Sparse Matrix-Tensor Factorizations by 200x - Supplementary Material

Evangelos E. Papalexakis[*] epapalex@cs.cmu.edu
Tom M. Mitchell [*] tom.mitchell@cmu.edu
Nicholas D. Sidiropoulos [†] nikos@ece.umn.edu
Christos Faloutsos [*] christos@cs.cmu.edu
Partha Pratim Talukdar [*] partha.talukdar@cs.cmu.edu
Brian Murphy [‡] brian.murphy@qub.ac.uk

This supplementary document serves as an appendix to the main manuscript. In Section 1, we provide an overview of one of our baselines, the Alternating Least Squares (ALS) algorithm, and we provide a modification thereof that yields significant speedup. In Section 2, we show how we can make Turbo-SMT, when using ALS, robust to missing values (in [2], the authors propose a version of their algorithm that deals with missing values). In Section 3, we compare Turbo-SMT to approaches that might consider compressing the tensor using a Tucker3 model. Finally, in Section 4, we showcase the generality of the CMTF framework, and consequently of Turbo-SMT, with an application to a time evolving social network, with side information.

In Table 1, we provide an overview of the symbols used.

Symbol	Description
CMTF	Coupled Matrix-Tensor Factorization
ALS	Alternating Least Squares
$x, \mathbf{x}, \mathbf{X}, \underline{\mathbf{X}}$	scalar, vector, matrix, tensor (respectively)
$\mathbf{A} \odot \mathbf{B}$	Khatri-rao product.
$\mathbf{A} \otimes \mathbf{B}$	Kronecker product.
$\mathbf{A} * \mathbf{B}$	Hadamard (elementwise) product.
$\mathbf{A}^\dagger$	Pseudoinverse of $\mathbf{A}$
$\|\mathbf{A}\|_F$	Frobenius norm of $\mathbf{A}$.
$\mathbf{a} \circ \mathbf{b} \circ \mathbf{c}$	$(\mathbf{a} \circ \mathbf{b} \circ \mathbf{c})(i,j,k) = \mathbf{a}(i)\mathbf{b}(j)\mathbf{c}(k)$
$\underline{\mathbf{X}}_{(i)}$	i-th mode unfolding of tensor $\underline{\mathbf{X}}$ (see [6]).

Table 1: Table of symbols

1 The Alternating Least Squares Algorithm

One of the most popular algorithms to solve PARAFAC (as introduced in Figure 1) is the so-called Alternating Least Squares (ALS); 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.

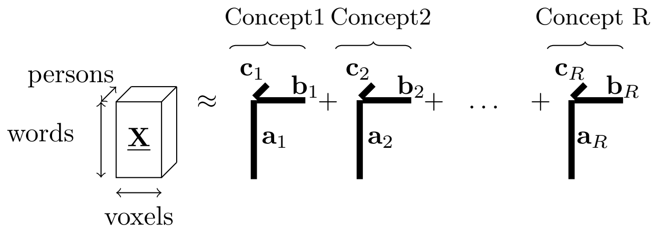

Figure 1: PARAFAC decomposition of a three-way tensor of a brain activity tensor as sum of F outer products (rank-one tensors), reminiscent of the rank-F singular value decomposition of a matrix. Each component corresponds to a **latent** concept of, e.g. "insects", "tools" and so on, a set of brain regions that are most active for that particular set of words, as well as groups of persons.

Solving CMTF using ALS follows the same strategy, only now, we have up to three additional matrices in our objective.

In Algorithm 1, we provide a detailed outline of the baseline algorithm, ALS for CMTF, also described in [1].

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

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

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

$$\mathbf{A} \odot \mathbf{B} = \begin{bmatrix} \mathbf{A}(:,1) \otimes \mathbf{B}(:,1) & \cdots & \mathbf{A}(:,F) \otimes \mathbf{B}(:,F) \end{bmatrix}$$

[*]Carnegie Mellon University
[†]University of Minnesota
[‡]Queen's University of Belfast

Algorithm 1: Baseline: Alternating Least Squares (ALS) Algorithm for CMTF

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

1.1 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 1.1. *We may do the following simplification to each pseudoinversion step of the ALS algorithm (Algorithm 1):*

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

Proof. For the Moore-Penrose pseudoinverse of the Khatri-Rao product, it holds that [4], [9]

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

Furthermore [4] $(\mathbf{A} \odot \mathbf{B})^T (\mathbf{A} \odot \mathbf{B}) = \mathbf{A}^T\mathbf{A} * \mathbf{B}^T\mathbf{B}$ 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 [5]

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

Putting things together, it follows:

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

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 $\mathbf{A}, \mathbf{B}$ and $\mathbf{M}$ are $I \times R$, $J \times R$ and $I \times I_2$, then computing the pseudoinverse naively would cost $O\left(R^2\left(IJ + I_2\right)\right)$, whereas our proposed method yields a cost of $O\left(R^2\left(I + J + I_2\right)\right)$ because of the fact that we are pseudoinverting only a *small* $R \times R$ matrix. We have to note here that in almost all practical scenarios $R \ll I, J, I_2$.

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

2 Accounting for 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 [10] provides a very comprehensive study on how to handle missing values for plain tensor decompositions.

We carefully ignore the missing values from the entire optimization procedure: Notice that is *not* the same as simply zeroing out all missing values, since 0 might have a valid physical interpretation. Specifically, we define a 'weight' tensor $\underline{\mathbf{W}}$ which has '0' in all coefficients where values are missing, and '1' everywhere else. Similarly, we introduce three weight matrices $\mathbf{W}_i$ for each of the coupled matrices $\mathbf{Y}_i$. Then, the optimization function of the CMTF model becomes

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

As we show in Algorithm 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

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

Table 2: Pseudoinversion speedup (100000 runs)

$$\min_{\mathbf{B}} \|\mathbf{W} * \left(\mathbf{X} - \mathbf{A}\mathbf{B}^T\right)\|_F^2 \tag{2.1}$$

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

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

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

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: $b = \frac{\mathbf{x}^T\mathbf{a}}{\|\mathbf{a}\|_2^2}$

We may, thus, solve this problem of Equation 2.1 using *element-wise coordinate descent*, where we update each coefficient of $\mathbf{B}$ iteratively, until convergence. Therefore, with the aforementioned derivation, we are able to modify our original algorithm in order to take missing values into account.

2.1 Experimental evaluation In order to measure resilience to missing values we define the *Signal-to-Noise Ratio* (SNR) as simply as SNR $= \frac{\|\underline{\mathbf{X}}_m\|_F^2}{\|\underline{\mathbf{X}}_m - \underline{\mathbf{X}}_0\|_F^2}$, where $\underline{\mathbf{X}}_m$ is the reconstructed tensor when a m fraction of the values are missing. In Figure 2, 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.

3 Comparison to Compression

One existing approach used for speeding up the PARAFAC decomposition is the so-called COMFAC algorithm [8]; COMFAC 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 [8], 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.

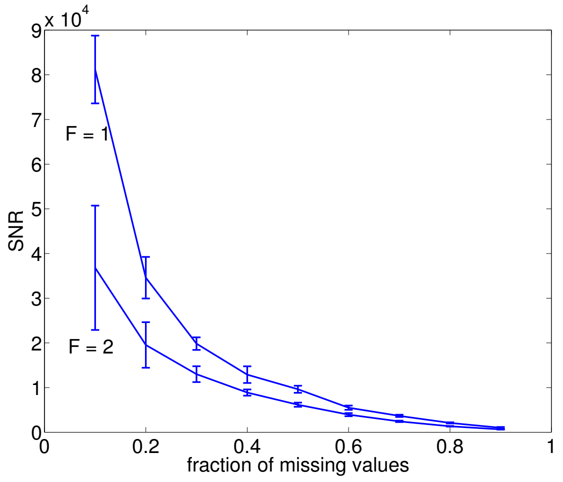

Figure 2: This Figure shows the Signal-to-Noise ratio (SNR)-as defined in the main text- as a function of the percentage of missing values. We can observe that, even for a fair amount of missing values, the SNR is quite high, signifying that Turbo-SMT is able to handle such ill-conditioned settings, with reasonable fidelity.

Consider the $I \times J \times K$ tensor $\underline{\mathbf{X}}$. Then, its $\{Q, R, P\}$ Tucker3 decomposition consists of a $P \times Q \times R$ core tensor, say, $\underline{\mathbf{G}}$ and three assorted, unitary, matrices $\mathbf{U}, \mathbf{V}, \mathbf{Z}$ 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:

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

Having a tensor $\underline{\mathbf{X}}$ coupled with matrices $\mathbf{Y}_i$, $i = 1\cdots 3$, we may first obtain the 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 $\mathbf{A}, \mathbf{D}$ to their original dimension (and accordingly for the rest of the factor matrices).

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

- Tucker3 is now a bottleneck; its computation (even though there exist memory efficient implementations in the literature [3], [7], which we use in our implementation) is very costly, compared to the simple sampling scheme that TURBO-SMT is using.
- 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 [1] produce dense factors. Therefore, this technique is prone to storage and interpretability issues.

We implemented this compression-based technique, using Tensor Toolbox's [3] memory efficient implementation of the Tucker3 decomposition. In Fig. 3, we illustrate the wall-clock time of this approach, compared to TURBO-SMT, on the entire BRAINQ dataset; we chose $s = 5$ for TURBO-SMT, and we chose $P = Q = R = 60$ for the compression-based technique. We observe that TURBO-SMT performs significantly better, while, additionally, producing sparse outputs.

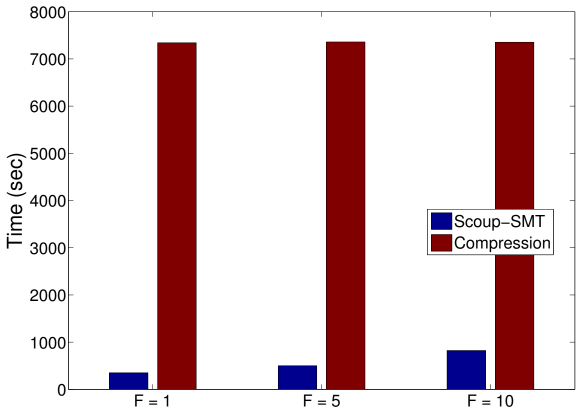

Figure 3: Comparison of TURBO-SMT and a compression-based technique, that uses the Tucker3 decomposition, as described in Sec. 3. TURBO-SMT significantly outperforms the alternative method.

4 Generality: Mining Social Networks with Additional Information

We have demonstrated the expressive power of TURBO-SMT for the BRAINQ dataset, but in this subsection, we stress the fact that the method is actually application independent and may be used in vastly different scenarios. To that end, we analyze a FACEBOOK dataset, introduced in [11][1]. This dataset consists of a $63890 \times 63890 \times 1847$ (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. 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.

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. 4 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 by using TURBO-SMT. The FACEBOOK dataset merits its own detailed analysis, however, here we include a small, preliminary result, due to space considerations.

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

References

[1] E. Acar, T.G. Kolda, and D.M. Dunlavy. All-at-once optimization for coupled matrix and tensor factorizations. *arXiv preprint arXiv:1105.3422*, 2011.

[2] E. Acar, G.E. Plopper, and B. Yener. Coupled analysis of in vitro and histology tissue samples to quantify

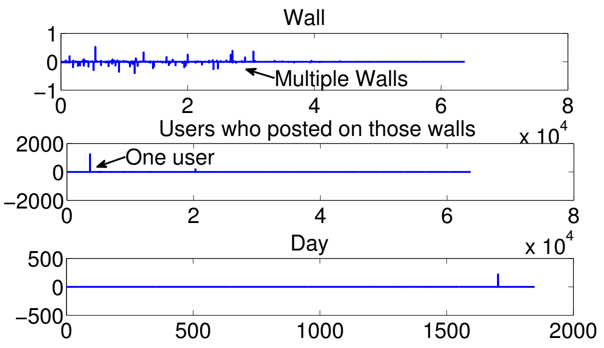

Figure 4: This is a pattern extracted using TURBO-SMT, which shows what appears to be a spammer on the FACEBOOK dataset: One person, posting to many different walls on a single day.

structure-function relationship. *PloS one*, 7(3):e32227, 2012.

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

[4] R. Bro. *Multi-way analysis in the food industry: models, algorithms, and applications.* PhD thesis, Københavns Universitet, 1998.

[5] C. Hung and T.L. Markham. The moore-penrose inverse of a partitioned matrix m= adbc. *Linear Algebra and its Applications*, 11(1):73–86, 1975.

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

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

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

[9] S. Liu and G. Trenkler. Hadamard, khatri-rao, kronecker and other matrix products. *International Journal of Information and Systems Sciences*, pages 160–177, 2008.

[10] G. Tomasi and R. Bro. Parafac and missing values. *Chemometrics and Intelligent Laboratory Systems*, 75(2):163–180, 2005.

[11] Bimal Viswanath, Alan Mislove, Meeyoung Cha, and Krishna P. Gummadi. On the evolution of user interaction in facebook. In *SIGCOMM Workshop on Social Networks*, 2009.