

# On low-rank updates to the singular value and Tucker decompositions

Michael J. O'Hara\*

## Abstract

The singular value decomposition is widely used in signal processing and data mining. Since the data often arrives in a stream, the problem of updating matrix decompositions under low-rank modification has been widely studied. Brand developed a technique in 2006 that has many advantages. However, the technique does not directly approximate the updated matrix, but rather its previous low-rank approximation added to the new update, which needs justification. Further, the technique is still too slow for large information processing problems. We show that the technique minimizes the change in error per update, so if the error is small initially it remains small. We show that an updating algorithm for large sparse matrices should be sub-linear in the matrix dimension in order to be practical for large problems, and demonstrate a simple modification to the original technique that meets the requirements.

We extend Brand's method to tensors, the multi-way generalization of matrices. The few published comparable techniques either focus on small-magnitude changes, are iterative, or have no published error properties. We show that the technique of Brand for updating the truncated singular value decomposition can be redesigned as a low-rank update scheme for the Tucker decomposition, with analogous run-time and error properties.

**Keywords:** Updating, truncated singular value decomposition, Tucker decomposition, low-rank approximation, tensor.

## 1 Introduction

In information processing, relationships between entities are often represented as a matrix. Matrix decompositions then tell us many useful things about the data [19, 9], with applications such as PageRank, latent semantic indexing, similarity and community-finding [10], and counting triangles [23], which is important in social network analysis.

To be useful in data mining, matrix decompositions need to be efficiently computed on a stream of data. The challenges posed by the limited memory and short processing times allowed by high-speed streaming environments have attracted interest [2]. In particular, counting triangles and estimating PageRank have been considered [7, 8, 16]. We expect the streaming problem on matrix decompositions to be represented as low-rank updates to the matrix, for instance, changing a few entries, adding or deleting a column or row, etc. Recom-

puting by, for instance, restarting the power method can be ineffective [16]; nonetheless other methods exist, and the subject has been studied for over thirty years, see, e.g., [5, 6, 11, 12, 20, 24].

Two recent techniques, building on their predecessors, stand out for their speed and accuracy, namely those by Brand [5] in 2006 and Koch and Lubich [12] in 2007 for updating the truncated singular value decomposition (truncated SVD). The essential difference between the Brand and Koch-Lubich technique is that the Brand technique will be accurate on large-magnitude changes, but become computationally-expensive if those changes are large-rank. The Koch and Lubich technique is efficient regardless of rank, but becomes inaccurate if the changes are large in magnitude. On the data we use for numerical experiments in this paper, where the changes are both low-rank and relatively small in magnitude, the techniques have comparable complexity and accuracy.

One concern with the Brand technique is that it does not directly approximate the updated matrix but rather a previous low-rank approximation added to the new update; the legitimacy of this substitution has not been explored. We show that the technique minimizes the change in error per update; therefore if the error is small initially, it remains small after updating. Another issue is that the technique is too slow for large information processing problems; an updating algorithm for large sparse matrices should be sub-linear in the matrix dimensions, and the Brand technique is linear in the matrix dimensions. We develop a simple modification to the original technique that is sub-linear in memory and run-time requirements.

Sometimes in information processing the relationships cannot be described with a matrix but rather require a multi-way array, otherwise known as tensors. This occurs when observations represent relationships between several entities at once. For instance, this chat user joined this chatroom and typed these key terms [1], or this author contributed to that conference in this year. Even to do Principle Components Analysis on time-evolving data requires a three-way array - the interaction between this variable and that variable at this time. Several studies applying tensor decompositions to information processing problems exist, e.g. [1, 3, 15].

---

\*Lawrence Livermore National Laboratory.

Streaming tensor algorithms have been shown useful in data mining applications [18, 21, 22].

We expect any update technique effective for the truncated SVD can be extended to the Tucker decomposition of tensors [13, 18, 22]. The challenge is extending the best technique in the best way. We extend the Brand technique to the Tucker decomposition, with analogous run-time and error properties.

Finally, we apply the algorithms to network flow data and perform experiments that yield insight into the effective application of the techniques.

## 2 Updating the truncated singular value decomposition

In the context of continuously evolving matrices  $A(t)$  and evolving low-rank approximation  $L(t)$ , Koch and Lubich [12] observed that it is simpler to minimize  $\|\dot{A}(t) - \dot{L}(t)\|$  (all norms are the Frobenius norm here, and the dot represents the time derivative) then to minimize  $\|A(t) - L(t)\|$ . If  $A(0) \approx L(0)$ , and  $\dot{A}(t) \approx \dot{L}(t)$ , then we expect  $A(t) \approx L(t)$  at least for a while.

Let the error of the approximation  $E(t)$  be  $A(t) - L(t)$ , then clearly

$$(2.1) \quad \dot{E}(t) = \dot{A}(t) - \dot{L}(t).$$

So we see that Koch and Lubich are really minimizing the change in error, as opposed to the error itself. In the discrete case, we have

$$(2.2) \quad \Delta E = \Delta A - \Delta L.$$

Let  $\Delta L = \bar{L} - L$  where  $L = USV^T$  and  $\bar{L} = \bar{U}\bar{S}\bar{V}^T$ . Then we write

$$(2.3) \quad \begin{aligned} \|\Delta E\| &= \|\Delta A - (\bar{U}\bar{S}\bar{V}^T - USV^T)\| \\ &= \|\bar{U}\bar{S}\bar{V}^T - (USV^T + \Delta A)\|. \end{aligned}$$

If we are requiring  $L(t)$  be a rank- $k$  approximation, then  $\|\Delta E\|$  is minimized provided  $\bar{U}\bar{S}\bar{V}^T$  is a best rank- $k$  approximation of  $USV^T + \Delta A$ . This is very significant. An update procedure based on the principle of minimizing change in error does not require the full representation of  $A$ , but merely its current low-rank approximation. The resulting updating problem is significantly easier. There are other justifications for this substitution as well [24].

The Brand technique efficiently computes the SVD of  $USV^T + XY^T$ , where  $X$  and  $Y$  are assumed low-rank. Thus we conclude that his method yields an optimal updating scheme, in the sense of minimizing the change in error per update, for low-rank updates. We review the scheme here, for the slightly-simplified case of rank-one updates.

The update technique starts with the observation:

$$(2.4) \quad USV^T + xy^T = [Ux] \begin{bmatrix} S & 0 \\ 0 & 1 \end{bmatrix} [Vy]^T.$$

The right-hand side of (2.4) looks like an SVD itself. However,  $[Ux]$  and  $[Vy]$  are not orthogonal. Define  $p = x - UU^T x$ , the projection of  $x$  onto the space orthogonal to the columns of  $U$ . Then it is easy to verify

$$(2.5) \quad [Ux] = [U\hat{p}] \begin{bmatrix} I & U^T x \\ 0 & \|p\| \end{bmatrix},$$

where  $\hat{p} = p/\|p\|$ . Let  $Q_1 = [U\hat{p}]$  and  $R_1$  be the second matrix in the product, so that  $[Ux] = Q_1 R_1$ . Evidently  $Q_1$  is orthogonal and of size  $n \times (k+1)$ , and  $R_1$  is of size only  $(k+1) \times (k+1)$ . Similarly, we can set  $q = y - VV^T y$ , and then

$$(2.6) \quad [Vy] = [V\hat{q}] \begin{bmatrix} I & V^T y \\ 0 & \|q\| \end{bmatrix}.$$

Let us then define  $Q_2$  and  $R_2$  correspondingly. Then we can write

$$(2.7) \quad USV^T + xy^T = Q_1 R_1 \begin{bmatrix} S & 0 \\ 0 & 1 \end{bmatrix} R_2^T Q_2^T,$$

where  $Q_1$  and  $Q_2$  are orthogonal, and the inner three matrices are of size  $(k+1) \times (k+1)$ .

Now, define

$$(2.8) \quad K = R_1 \begin{bmatrix} S & 0 \\ 0 & 1 \end{bmatrix} R_2^T.$$

Notice that  $K$  is only  $(k+1) \times (k+1)$ , so it is inexpensive to compute the SVD  $K = \Psi \Sigma \Phi^T$  if  $k$  is small. Then the updated SVD is

$$(2.9) \quad USV^T + xy^T = (Q_1 \Psi) \Sigma (Q_2 \Phi)^T.$$

Finally, we can truncate the least singular value to obtain the best rank- $k$  approximation.

A bad situation occurs when  $U^T X = 0$  and  $V^T Y = 0$ . For then, unless  $\|XY^T\| > S_{min}$  where  $S_{min}$  is the least diagonal entry of  $S$ , the low-rank approximation will be unaffected by the update. We can get around this problem by keeping a basis for the error subspace [20], though this may require a great deal of storage if  $A$  is large and sparse.

**2.1 DOWNDATING** There are two ways that we can downdate, which we might characterize as “hard” and “soft” downdates. A hard downdate involves including only observations within a moving window. Downdates then are the same as updates but with a negative

sign. A soft downdate is to decay the edge weights exponentially. In that case, we update by finding the best rank- $k$  approximation to  $(1 - \alpha)USV^T + \alpha y^T$  for some small  $\alpha$ . The “half-life” of the edges is then  $1/\alpha$ . Soft downdates are more efficient and appealing intuitively because old observations decay smoothly in importance with age, so this is what we use in the numerical experiments in this paper.

**2.2 Run-time requirements for updating in large problems** Suppose  $A$  is  $m \times n$ , where  $n > m$ . The update process described requires a constant number of matrix multiplies and vector operations, the most expensive of which is multiplying a  $n \times (k+1)$  matrix by a  $(k+1) \times (k+1)$  matrix. For a small fixed  $k$ , the work per update is  $O(n)$ . This is a significant improvement over the non-updating computation of the singular value decomposition, which is iterative and uses matrix-vector multiplies as the core operation. The work required for a sparse matrix multiply is proportional to the number of non-zero entries, which grows faster than  $n$  in information processing graphs [17].

Unfortunately,  $O(n)$  is also the minimum work for any exact update procedure, because each of the entries in the singular vectors need to be changed upon each update. If the number of observations is  $t$ , then the overall work is  $O(nt)$ . But  $n$  and  $t$  may both be in the billions, as in the PageRank matrix [9], so the total computation is impractical with current technology.

There is also the issue of memory requirements. Large graphs in information processing may be stored on distributed systems, but we would not like to have to access memory on all the distributed systems for each update. We prefer the update procedure to have both run-time and memory requirements that are  $O(\log n)$ .

Our central observation for speeding up the Brand technique is that we can compress  $U$  and  $V$  by keeping only  $C \log n$  of the largest-magnitude non-zero entries in each column, for some constant  $C$ , and using Gram-Schmidt to keep the columns orthogonal. This is an orthogonal variant to the sparsification routine suggested by Zhang et al. [25].

The first time this is done, we have to sort the columns which requires time  $O(n \log n)$ . However, assuming the update vectors  $x$  and  $y$  very sparse, the update then only costs  $O(\log n)$  for fixed  $k$ . Further, each column of  $U$  and  $V$  now has at most  $kC \log n$  non-zero entries, so we can quickly re-sort the columns and again keep only the  $C \log n$  largest-magnitude entries. Thus, subsequent updates are  $O(\log n)$  in their entirety, for a fixed  $k$ .

While extremely simple, we view this idea as important for making updating reasonable for large prob-

lems. It succeeds provided that the singular vectors are amenable to approximation with sparse vectors.

Numerical experiments on network traffic data, discussed in more detail in Section 4, show that sparsifying the singular vectors does degrade the accuracy, but not catastrophically (Figure 4) on this data, and speeds up the computation (Figure 3).

### 3 Updating the Tucker decomposition

The generalization of the SVD to higher dimensions is called the higher-order SVD (HOSVD) [9]. It is calculated by finding the SVD of all the unfoldings (rearrangements of the tensor into a matrix), and applying the resulting transformations to the tensor. Unfortunately, unlike for matrices, truncations of the HOSVD do not lead to optimal low-rank approximations.

Tucker decompositions, which we take to mean optimal low-rank tensor factorizations, are usually computed with an iterative alternating least-squares (ALS) technique. For instance, in a three-mode tensor, we can use some guess for the factor matrices for two of the modes, and then solve for the third mode as a linear least squares problem. Now we can assume the third mode and solve for one of the first two, etc. This process is assumed to converge to a local minimum solution.

The HOSVD calculation or the ALS process can be modified for a streaming environment in obvious ways. We can use SVD updating to update the HOSVD, and we can use the previous solution from ALS to initialize a new round of ALS - further, we can choose to use only one round of ALS to compute updates. Finally, the SVD is used as a subroutine and we can use efficient SVD updating in the obvious way to accelerate these computations. These ideas are explored in several papers [22, 21, 18]. The downside is that these techniques have no error bounds, nor, when relevant, convergence properties. A more sophisticated approach is taken by Koch and Lubich [13], who generalize their results for the SVD nicely to the Tucker decomposition. We show how the concepts developed by Brand for low-rank modifications to the SVD generalize nicely to Tucker decomposition.

**3.1 Tensor Notation** We briefly review our notation. Vectors are lowercase, matrices are uppercase, and tensors are script uppercase. Each dimension will usually be referred to as a *mode*, e.g., modes one, two, and three.

We define the *outer product* of three (column) vectors  $x$ ,  $y$ , and  $z$  as the three-way tensor with entries  $x_i y_j z_k$  in the  $(i, j, k)$  position. We denote this outer product as  $x \circ y \circ z$ . Notice that  $x \circ y = xy^T$ .

We define the *rank* of a tensor to be the minimum

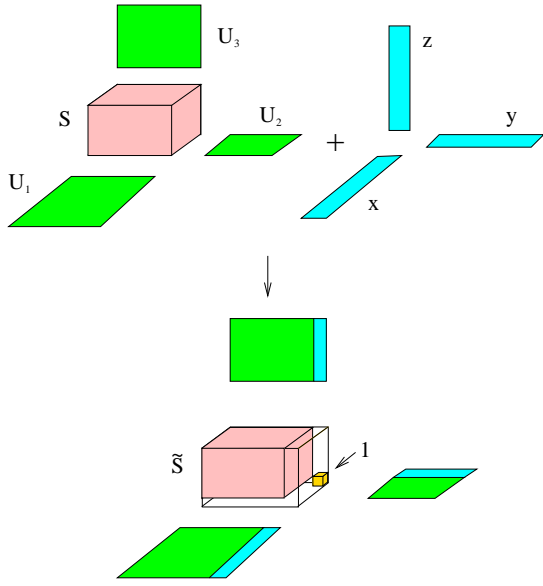


Figure 1: Illustration of how we add a rank-one term to a Tucker decomposition. The new factorization needs to be orthogonalized and truncated.

number of outer products that have to be added together to give you that tensor.

Columns in each dimension are called *fibers*. Recall that matrix multiplication  $XY$  can be thought of as transforming each column of  $Y$  by the matrix  $X$ . So a matrix-tensor product in a given mode is the transformation of the fibers in that mode by the matrix, and is denoted  $\times_i$ . For instance

$$(3.10) \quad (\mathcal{A} \times_1 U)_{ijk} = \sum_m u_{im} \mathcal{A}_{mjk} .$$

Notice if  $\mathcal{A}$  is a two-way tensor (a matrix), then  $\mathcal{A} \times_1 U = U\mathcal{A}$  and  $\mathcal{A} \times_2 U = \mathcal{A}U^T$ .

The Tucker decomposition on a three-way tensor  $\mathcal{A}$  is

$$(3.11) \quad \mathcal{A} \approx \mathcal{S} \times_1 U_1 \times_2 U_2 \times_3 U_3 ,$$

where the *core tensor*  $\mathcal{S}$  is a dense tensor of much smaller dimension than  $\mathcal{A}$ , and the transformation matrices  $U_i$  are orthogonal.

**3.2 Extending the Brand technique to the Tucker decomposition** The following analysis for tensors is analogous as that for matrices in Section 2, but we repeat it because the notation is different.

In the context of continuously evolving tensors  $\mathcal{A}(t)$  and evolving Tucker decompositions  $\mathcal{T}(t)$ , Koch and Lubich [13] observed that it is simpler to minimize

$\|\dot{\mathcal{A}}(t) - \dot{\mathcal{T}}(t)\|$  then to minimize  $\|\mathcal{A}(t) - \mathcal{T}(t)\|$ . If  $\mathcal{A}(0) \approx \mathcal{T}(0)$ , and  $\dot{\mathcal{A}}(t) \approx \dot{\mathcal{T}}(t)$ , then we expect  $\mathcal{A}(t) \approx \mathcal{T}(t)$  at least for a while.

Let the error of the approximation  $\mathcal{E}(t)$  be  $\mathcal{A}(t) - \mathcal{T}(t)$ , then clearly

$$(3.12) \quad \dot{\mathcal{E}}(t) = \dot{\mathcal{A}}(t) - \dot{\mathcal{T}}(t) .$$

So we see that Koch and Lubich are really minimizing the change in error, as opposed to the error itself. In the discrete case, we have

$$(3.13) \quad \Delta\mathcal{E} = \Delta\mathcal{A} - \Delta\mathcal{T} .$$

Let  $\Delta\mathcal{T} = \bar{\mathcal{T}} - \mathcal{T}$ . Then we write

$$(3.14) \quad \begin{aligned} \|\Delta\mathcal{E}\| &= \|\Delta\mathcal{A} - (\bar{\mathcal{T}} - \mathcal{T})\| \\ &= \|\bar{\mathcal{T}} - (\mathcal{T} + \Delta\mathcal{A})\| . \end{aligned}$$

As before, if we are requiring  $\mathcal{T}(t)$  to be a  $k_1 \times k_2 \times k_3$  Tucker decomposition, then  $\|\Delta\mathcal{E}\|$  is minimized provided  $\bar{\mathcal{T}}$  is a best  $k_1 \times k_2 \times k_3$  Tucker decomposition of  $\mathcal{T} + \Delta\mathcal{A}$ .

Now, let us compute  $\bar{\mathcal{T}}$  for the case that  $\Delta\mathcal{A}$  is rank-one. Let us take  $\Delta\mathcal{A} = x \circ y \circ z$ . We note that the techniques generalize to higher-rank updates in the same way the Brand technique generalizes to higher-rank updates on matrices, but the notation becomes more complex. So we want to find the  $k_1 \times k_2 \times k_3$  Tucker decomposition of

$$(3.15) \quad \mathcal{S} \times_1 U_1 \times_2 U_2 \times_3 U_3 + x \circ y \circ z .$$

As illustrated in Figure 1, we expand the core tensor  $\mathcal{S}$  to the  $(k_1+1) \times (k_2+1) \times (k_3+1)$  tensor  $\tilde{\mathcal{S}}$ . We fill out the new entries with zeros, except set  $(k_1+1, k_2+1, k_3+1)$  to one. Then we can write

$$(3.16) \quad \begin{aligned} \mathcal{S} \times_1 U_1 \times_2 U_2 \times_3 U_3 + x \circ y \circ z &= \\ \tilde{\mathcal{S}} \times_1 [U_1 \ x] \times_2 [U_2 \ y] \times_3 [U_3 \ z] . \end{aligned}$$

We factor  $[U_1 \ x] = Q_1 R_1$  etc, as before, and set

$$(3.17) \quad \mathcal{K} = \tilde{\mathcal{S}} \times_1 R_1 \times_2 R_2 \times_3 R_3 .$$

This is a  $(k_1+1) \times (k_2+1) \times (k_3+1)$  dense tensor. We can compute the  $k_1 \times k_2 \times k_3$  Tucker decomposition of it quickly if  $k_1 k_2 k_3$  is small, and we get

$$(3.18) \quad \mathcal{K} \approx \mathcal{C} \times_1 \Psi_1 \times_2 \Psi_2 \times_3 \Psi_3 .$$

So our updated Tucker decomposition is:

$$(3.19) \quad \mathcal{T} + x \circ y \circ z \approx \mathcal{C} \times_1 (Q_1 \Psi_1) \times_2 (Q_2 \Psi_2) \times_3 (Q_3 \Psi_3) .$$

We note that the same sparsification technique that we applied to the Brand technique for updating the SVD can be applied to this technique as well.

## 4 Numerical Experiments

We wrote Matlab functions to implement the described algorithms, using the tensor toolbox[4]. We applied the techniques to Lawrence Livermore National Laboratory institutional datasets of network traffic.

For the singular value decomposition experiments, we used the first million netflows from a conference dataset. We used most of the netflows, holding out a variable number of flows at the end, to build a large matrix to update, using the source and destination IP addresses as the rows and columns and the number of netflows seen as the matrix entries. We then used the held out netflows as rank-one updates to the matrix. To incorporate downdating, we decayed edge weights exponentially as new observations were added, with a downdate half-life of ten thousand observations.

Figure 2 shows the benefits of aggregating a few observations together before running the update routine. Here, we read in the first 970,000 observations, and then used the next thousand for updates, repeating the experiment taking one, two, three, etc. observations at a time per update. After 970,000 observations the matrix has 212,968 non-zero elements and dimensions  $131,731 \times 131,734$ . At this scale, it takes about three seconds to run Matlab’s “svds” routine, on the computer<sup>1</sup> used for the experiments. We see the work for larger-rank updates grows asymptotically faster than the linear benefit of aggregation, so there is an optimum. The optimum in the experiment was rank-three updates for the full Brand technique and rank-five for the sparsified version.

Figure 3 compares the run-time of updating between the original Brand technique and the sparsified Brand technique with  $C = 5$ , and overlays the time required to run Matlab’s “svds”. To vary the size of the matrix, we vary the number of observations used to construct the matrix to be updated. Then we take the next thousand observations, using three observations per update, and average over the update times. When the initial matrix is constructed from a million observations, we can update 261 observations with the sparsified Brand technique in the time it takes to run “svds” once. Note “svds” internally uses a highly-optimized fortran routine, whereas the update routine is written in high-level Matlab instructions.

Figure 4 shows the error of the Brand updated approximation compared to the accelerated version where the small entries in the singular vectors are dropped. We see degradation but the accelerated version is still more accurate than not updating at all. Here, we read the first 970,000 observations out of the netflow file, and

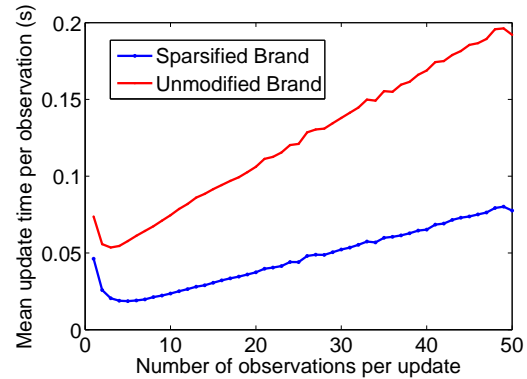


Figure 2: By combining a few observations at a time to form low-rank (instead of rank-one) updates, we can speed up the rate at which we process observations. The optimum is rank-three for the full Brand technique and rank-five for the sparsified version.

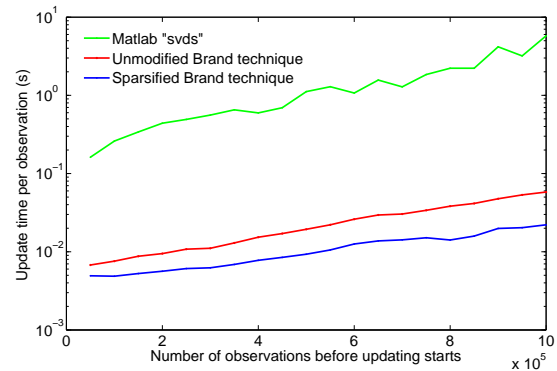


Figure 3: Time required to process updates using the Brand technique and the sparsified Brand technique, compared to the run-time of Matlab’s “svds” routine, as a function of matrix size.

<sup>1</sup>MacBook Pro laptop with dual-core 2.5GHz CPU and 3GB RAM.

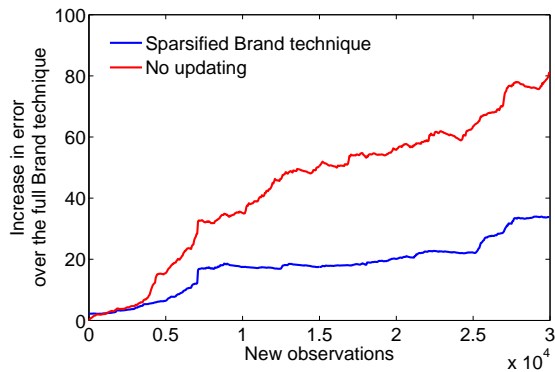


Figure 4: Frobenius norm of the error of the “sparsified” Brand technique, compared to the error of the non-updated approximation. Both have been subtracted from the original Brand technique. There is an instantaneous degradation due to the sparsification, but over time it is better to sparsify and update than to not update at all. For comparison, the Frobenius norm of  $A$  after processing the updates is 1230.

used the last 30,000 for updating. We took five observations per update, and the error was computed every ten updates, using the formula in Kolda and Bader’s review [14, p. 478].

For the tensor decompositions, which are much slower to compute, we used a much smaller dataset of 13,355 netflows from a smaller conference dataset collected over several days. We used source and destination IP addresses for two of the modes, and divided times into one-hour intervals for the remaining mode. The resulting tensor had dimensions  $67 \times 1857 \times 1958$  with 2610 non-zero elements.

Figure 5 illustrates that it is important to use a low-rank update technique as opposed to an incremental update technique for sufficiently-large updates. Here we use all but the last ten observations to build the original tensor and  $4 \times 4 \times 4$  Tucker decomposition, and used the last ten observations multiplied by a scaling factor for updates. We repeated the experiment using different scaling factors. For scaling factors of several hundred we see that the error of the incremental technique strays, but that the error of the low-rank method is contained.

We note that our implementation of our technique was about twelve times faster than our implementation of the Koch and Lubich technique, and several dozen times faster than one round of ALS as implemented in the tensor toolbox.

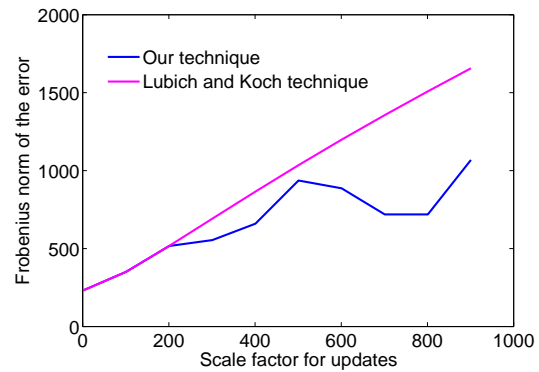


Figure 5: Error of our technique and of the Koch and Lubich technique after ten updates, for different scalings of the update magnitudes. As the updates are made large, it becomes important to use a low-rank update method instead of an incremental update method.

## 5 Significance and Impact

We have revealed that the Brand technique [5] is the natural analog to the Koch and Lubich technique [12] for updating the truncated singular value decomposition, but for low-rank updates as opposed to incremental updates. We have extended the Brand technique to low-rank updates of the Tucker decomposition. We determined that  $O(n)$  update procedures are too slow for large-scale information processing problems, and identified and implemented a candidate solution involving replacing the singular vectors with approximate sparse representations.

## 6 Acknowledgements

The author would like to thank Michael Last, Tammy Kolda, Joe McCloskey, Dianne O’Leary, and the anonymous referees for helpful suggestions and feedback.

## References

- [1] E. Acar, S. Camtepe, and B. Yener. Collective sampling and analysis of higher order tensors for chatroom communications. In *Proceedings of the IEEE International Conference on Intelligence and Security Informatics*, 2006.
- [2] C. C. Aggarwal, editor. *Data streams: models and algorithms*. Springer, New York, NY, 2007.
- [3] B. W. Bader, M. W. Berry, and M. Browne. Discussion tracking in Enron email using PARAFAC. In *Survey of Text Mining II: Clustering, Classification, and Retrieval*, pages 147–163. Springer, 2007.
- [4] B. W. Bader and T. G. Kolda.

- Matlab tensor toolbox version 2.3. <http://csmr.ca.sandia.gov/tgkolda/TensorToolbox>.
- [5] M. Brand. Fast low-rank modifications of the thin singular value decomposition. *Linear algebra and its applications*, 415(1):20–30, May 2006.
- [6] J. Bunch, C. Nielsen, and D. Sorensen. Rank-one modification of the symmetric eigenproblem. *Numerische Mathematik*, 31:31–48, 1978.
- [7] S. Buriol, G. Frahling, S. Leonardi, A. Marchetti-Spaccamela, and C. Sohler. Counting triangles in data streams. In *ACM Symposium on Principles of Database Systems*, 2006.
- [8] A. Das Sarma, S. Gollapudi, and R. Panigrahy. Estimating PageRank on graph streams. In *ACM Symposium on Principles of Database Systems*, 2008.
- [9] L. Elden. *Matrix methods in data mining and pattern recognition*. SIAM, Philadelphia, PA, 2007.
- [10] F. Fouss, A. Pirotte, J.-M. Renders, and M. Saerens. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Trans. Knowledge and Data Engineering*, 19(3):355–369, 2007.
- [11] G. H. Golub and C. F. V. Loan. *Matrix Computations*. The Johns Hopkins University Press, 3rd edition, 1996.
- [12] O. Koch and C. Lubich. Dynamical low-rank approximation. *SIAM Journal on Matrix Analysis and Applications*, 29(2):434–454, 2007.
- [13] O. Koch and C. Lubich. Dynamical low rank approximation of tensors. In *13th International congress on computational and applied mathematics*, 2008.
- [14] T. Kolda and B. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3), 2009.
- [15] T. Kolda, B. Bader, and J. Kenny. Higher-order web link analysis using multilinear algebra. In *Proceedings of the 5th IEEE International Conference on Data Mining*, pages 242–249. IEEE Computer Society, 2005.
- [16] A. N. Langville and C. D. Meyer. Updating Markov chains with an eye on Google’s PageRank. *SIAM J. Matrix Anal. Appl.*, 27(4):968–987, 2006.
- [17] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: Densification laws, shrinking diameters and possible explanations. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2005.
- [18] X. Li, W. Hu, Z. Zhang, X. Zhang, and G. Luo. Robust visual tracking based on incremental tensor subspace learning. In *Proceedings of the IEEE International Conference on Computer Vision*, 2007.
- [19] D. Skillicorn. *Understanding complex datasets: data mining with matrix decompositions*. Chapman and Hall, Boca Raton, FL, 2007.
- [20] G. W. Stewart. An updating algorithm for subspace tracking. *IEEE Trans. Signal Processing*, 40:1535–1541, 1992.
- [21] J. Sun, S. Papadimitriou, and P. Yu. Window-based tensor analysis on high-dimensional and multi-aspect streams. In *Proceedings of the 6th IEEE International Conference on Data Mining*, 2006.
- [22] J. Sun, D. Tao, and C. Faloutsos. Beyond streams and graphs: dynamic tensor analysis. In *Proceedings of the 12th ACM SIGDDK international conference on knowledge discovery and data mining*, 2006.
- [23] C. E. Tsourakakis. Fast counting of triangles in large real networks without counting: algorithms and laws. In *IEEE International Conference on Data Mining*, 2008.
- [24] H. Zha and H. D. Simon. On updating problems in latent semantic indexing. *SIAM Journal on Scientific Computing*, 21(2):782–791, 1999.
- [25] Z. Zhang, H. Zha, and H. D. Simon. Low-rank approximations with sparse factors i: basic algorithms and error analysis. *SIAM Journal on Matrix Analysis and Applications*, 23(3):706–727, 2002.