Beyond Streams and Graphs: Dynamic Tensor Analysis

Jimeng Sun† Dacheng Tao‡ Christos Faloutsos†
† Computer Science Department, Carnegie Mellon University, Pittsburgh,USA
‡ School of Computer Science and Information Systems, Birkbeck College, University of London,UK
{jimeng,christos}@cs.cmu.edu, dacheng@dcs.bbk.ac.uk

ABSTRACT

How do we find patterns in author-keyword associations, evolving over time? Or in DataCubes, with product-branch-customer sales information? Matrix decompositions, like principal component analysis (PCA) and variants, are invaluable tools for mining, dimensionality reduction, feature selection, rule identification in numerous settings like streaming data, text, graphs, social networks and many more. However, they have only two orders, like author and keyword, in the above example.

We propose to envision such higher order data as *tensors*, and tap the vast literature on the topic. However, these methods do not necessarily scale up, let alone operate on semi-infinite streams. Thus, we introduce the *dynamic tensor analysis* (DTA) method, and its variants. DTA provides a compact summary for high-order and high-dimensional data, and it also reveals the hidden correlations. Algorithmically, we designed DTA very carefully so that it is (a) *scalable*, (b) *space efficient* (it does not need to store the past) and (c) *fully automatic* with no need for user defined parameters. Moreover, we propose STA, a streaming tensor analysis method, which provides a fast, streaming approximation to DTA.

We implemented all our methods, and applied them in two real settings, namely, anomaly detection and multi-way latent semantic indexing. We used two real, large datasets, one on network flow data (100GB over 1 month) and one from DBLP (200MB over 25 years). Our experiments show that our methods are fast, accurate and that they find interesting patterns and outliers on the real datasets.

1. INTRODUCTION

Given a keyword-author-timestamp-conference bibliography, how can we find patterns and latent concepts? Given Internet traffic data (who sends packets to whom, on what port, and when), how can we find anomalies, patterns and summaries? Anomalies could be, e.g., port-scanners, patterns could be of the form "workstations are down on weekends, while servers spike at Fridays for backups". Summaries like the above one are useful to give us an idea what is the past (which is probably the norm), so that we can spot deviations from it in the future.

Matrices and matrix operations like SVD/PCA have played a vital role in finding patterns when the dataset is "2-dimensional", and can thus be represented as a matrix. Important applications of this view point include numerous settings like:

1) **information retrieval**, where the data can be turned into a document-term matrix, and then apply LSI [9, 23]; 2) **market basket analysis**, with customer-products matrices, where we can apply association rules [2] or "Ratio Rules" [20]; 3) **the web**, where both rows and columns are pages, and links correspond to edges between them; then we can apply HITS [19] or pageRank [3] to find hubs, authorities and influential nodes; all of them are identical or closely related to eigen analysis or derivatives; 4) **social networks**, and in fact, *any* graph (with un-labelled edges): people are rows and columns; edges again correspond to non-zero entries in the adjacency matrix. The network value of a customer [13] has close ties to the first eigenvector; graph partitioning [18] is often done through matrix algebra (e.g. spectral clustering [16]); 5) **streams and co-evolving sequences** can also be envisioned as matrices: each data source (sensor) corresponds to a row, and each time-tick to a column. Then we can do multivariate analysis or SVD [24],"sketches" and random projections [14] to find patterns and outliers.

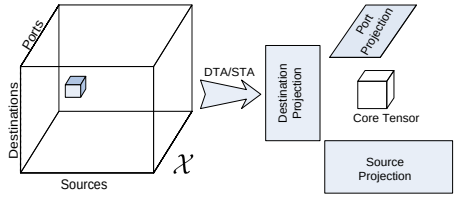
The need for tensors: Powerful as they may be, matrix-based tools can handle neither of the two problems we stated in the beginning. The crux is that matrices have only two "dimensions" (e.g., "customers" and "products"), while we may often need more, like "authors", "keywords", "timestamps", "conferences". This is exactly what a *tensor* is, and of course, a tensor is a generalization of a matrix (and of a vector, and of a scalar). We propose to envision all such problems as tensor problems, to use the vast literature of tensors to our benefit, and to introduce new tensor analysis tools, tailored for streaming applications. Using tensors, we can attack an even wider range of problems, that matrices can not even touch. For example, 1) Rich, time-evolving network traffic data, as mentioned earlier: we have tensors of order $M = 3$, with modes "source-IP", "destination-IP" and "port" over time. 2) Labeled graphs and social networks: suppose that we have different types of edges in a social network (eg., who-called-whom, who-likes-whom, who-emailed-whom, who-borrowed-money-from-whom). In that case, we have a 3rd order tensor, with edge-type being the 3rd mode. Over time, we have multiple 3rd order tensors, which is still within the reach of our upcoming tools. 3) Microarray data, with gene expressions evolving over time [31]. Here we have genes-proteins over time, and we record the

OLAP	this paper	tensor literature
dimensionality	order	order/mode
dimension	mode	order/mode
attribute value	dimension	dimension

Table 1: Terminology correspondence

expression level: a series of 2nd order tensors. 4) All OLAP and DataCube applications: customer-product-branch sales data is a 3rd order tensor; and so is patient-diagnoses-drug treatment data.

Motivating example: Let us consider the network monitoring example. Here we have network flows arriving very fast and continuously through routers, where each flow consists of source IP, destination IP, port number and the number of packets. How to monitor the dynamic traffic behavior? How to identify anomalies which can signify a potential intrusion, or worm propagation, or an attack? What are the correlations across the various sources, destinations and ports?



(a) original Tensor (b) DTA result

Figure 1: 3rd order tensor of network flow data

Therefore, from the **data model aspect**, we propose to use a more general and expressive model *tensor sequences*. For the network flow example, the 3rd order tensor for a given time period has three modes: source, destination and port, which can be viewed as a 3D data cube (see Figure 1(a)). An entry (i, j, k) in that tensor (like the small blue cube in Figure 1(a)) has the number of packets from the corresponding source (i) to the destination j through port k, during the given time period. The dynamic aspect comes from the fact that new tensors are arriving continuously over time.

From the **algorithmic aspect**, we propose the *dynamic* and the *streaming tensor analysis* (DTA and STA) to summarize the original tensors into smaller "*core*" tensors and the "*projection matrices*" (one for each mode), as shown in Figure 1(b). The projection matrices can be updated incrementally over time when a new tensor arrives, and contain valuable information about which, eg., source IP addresses are correlated with which destination IP addresses and destination ports, over time.

Finally, from the **application aspect**, we illustrate the anomaly detection process and *multi-way LSI* through DTA/STA. For the former, our method found suspicious Internet activity in a real trace, and it also found the vast majority of injected anomalies (see Section 6). For the latter, we found natural groups of authors, keywords and time intervals, in the DBLP bibliography subset.

Our contributions:

- The introduction of *tensors* to streaming analysis.
- Two new fast, incremental tools, the *dynamic* and the *streaming tensor analysis* (DTA and STA).
- Systematic evaluation on large, real datasets, where our tools found patterns and anomalies, and provided large speed-ups over competitors.

The rest of the paper is organized as the following: Section 2 introduces the necessary background. Then Section 3 formally defines the problem and gives an overview of the methods. Section 4 presents the key methods dynamic tensor analysis (DTA) and streaming tensor analysis (STA). Using DTA or STA, Section 5 illustrates two important applications: anomaly detection and multi-way latent semantics indexing. In Section 6 and Section 7 we extensively evaluate all the proposed methods using two real datasets. Section 8 discusses the related work. Finally we conclude in Section 9.

Symbol	Description
$\mathbf{v}$	a vector (lower-case bold)
$\mathbf{v}(i)$	the i-element of vector $\mathbf{v}$
$\mathbf{A}$	a matrix (upper-case bold)
$\mathbf{A}^T$	the transpose of $\mathbf{A}$
$\mathbf{A}_i\|_{i=1}^n$	a sequence of N matrices $\mathbf{A}_1, \ldots, \mathbf{A}_n$
$\mathbf{A}(i,j)$	the entry (i,j) of $\mathbf{A}$
$\mathbf{A}(i,:)$ or $\mathbf{A}(:,i)$	i-th row or column of $\mathbf{A}$
$\mathcal{A}$	a tensor (calligraphic style)
$\mathcal{A}(i_1, \ldots, i_M)$	the element of $\mathcal{X}$ with index $(i_1, \ldots, i_M)$
M	the order of the tensor
N_i	the dimensionality of the ith mode ($1 \leq i \leq M$)

Table 2: Description of notation.

2. BACKGROUND

Here we briefly give the main concepts and terms from principal component analysis (PCA), and from tensor algebra, also known as multilinear analysis.

2.1 Principal Component Analysis

PCA, as shown in Figure 2, finds the best linear projections of a set of high dimensional points to minimize least-squares cost. More formally, given n points represented as row vectors $\mathbf{x}_i|_{i=1}^n \in \mathbb{R}^N$ in an N dimensional space, PCA computes n points $\mathbf{y}_i|_{i=1}^n \in \mathbb{R}^R$ ($R \ll N$) in a lower dimensional space and the projection matrix $\mathbf{U} \in \mathbb{R}^{N \times R}$ such that the least-squares cost $e = \sum_{i=1}^n \|\mathbf{x}_i - \mathbf{y}_i \mathbf{U}^T\|_2^2$ is minimized[1].

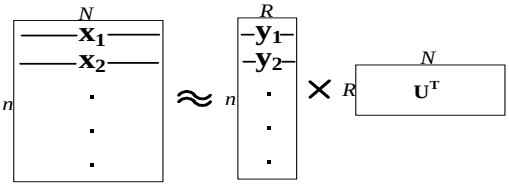


Figure 2: PCA projects the N-D vector $\mathbf{x}_i$s into R-D vector $\mathbf{y}_i$s and U is the projection matrix.

The solution of PCA can be computed efficiently by diagonalizing the covariance matrix of $\mathbf{x}_i|_{i=1}^n$. Alternatively, if the rows are zero mean, then PCA is computed by the Singular Value Decomposition (SVD): if the SVD of $\mathbf{X}$ is $\mathbf{X} = \mathbf{U}_{svd} \times \mathbf{\Sigma}_{svd} \times \mathbf{V}_{svd}{}^T$, then our $\mathbf{Y} = \mathbf{U}_{svd} \times \mathbf{\Sigma}_{svd}$ and $\mathbf{U} = \mathbf{V}_{svd}$

The intuition behind PCA is the following: if $\mathbf{X}$ is the author-keyword matrix of the DBLP dataset, then matrix $\mathbf{Y}$ is roughly the author – research-Area matrix, and matrix $\mathbf{U}$ is the keyword – research-Area matrix.

2.2 Multilinear Analysis

[1]Both $\mathbf{x}$ and $\mathbf{y}$ are row vectors.

As mentioned, a tensor of order M closely resembles a Data Cube with M dimensions. Formally, we write an Mth order tensor $\mathcal{X} \in \mathbb{R}^{N_1 \times \cdots \times N_M}$ as $\mathcal{X}_{[N_1,\ldots,N_M]}$, where N_i $(1 \le i \le M)$ is the *dimensionality* of the ith mode ("dimension" in OLAP terminology). For brevity, we often omit the subscript $[N_1, \ldots, N_M]$.

We will also follow the typical conventions, and denote matrices with upper case bold letters (e.g., $\mathbf{U}$) row vectors with lower-case bold letters (e.g., $\mathbf{x}$, scalars with lower-case normal font (e.g., n), and tensors with calligraphic font (e.g., $\mathcal{X}$. From the tensor literature we need the following definitions:

DEFINITION 1 (MATRICIZING OR MATRIX UNFOLDING). *The mode-d matricizing or matrix unfolding of an Mth order tensor $\mathcal{X} \in \mathbb{R}^{N_1 \times \cdots \times N_M}$ are vectors in $\mathbb{R}^{N_d}$ obtained by keeping index d fixed and varying the other indices. Therefore, the mode-d matricizing $\mathbf{X}_{(d)}$ is in $\mathbb{R}^{(\prod_{i \ne d} N_i) \times N_d}$.*

The operation of mode-d matricizing $\mathcal{X}$ is denoted as unfold($\mathcal{X}$,d)=$\mathbf{X}_{(d)}$. Similarly, the inverse operation is denoted as fold($\mathbf{X}_{(d)}$). In particular, we have $\mathcal{X}$ = fold(unfold($\mathcal{X}, d$)). Figure 3 shows an example of mode-1 matricizing of a 3rd order tensor $\mathcal{X} \in \mathbb{R}^{N_1 \times N_2 \times N_3}$ to the $(N_2 \times N_3) \times N_1$-matrix $\mathbf{X}_{(1)}$. Note that the shaded area of $\mathbf{X}_{(1)}$ in Figure 3 the slice of the 3rd mode along the 2nd dimension.

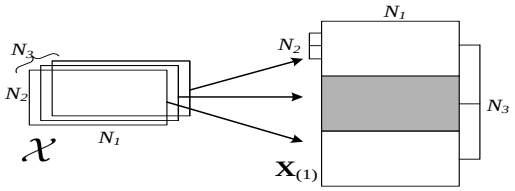


Figure 3: 3rd order tensor $\mathcal{X} \in \mathbb{R}^{N_1 \times N_2 \times N_3}$ is matricized along mode-1 to a matrix $\mathbf{X}_{(1)} \in \mathbb{R}^{(N_2 \times N_3) \times N_1}$. The shaded area is the slice of the 3rd mode along the 2nd dimension.

DEFINITION 2 (MODE PRODUCT). *The mode product $\mathcal{X} \times_d \mathbf{U}$ of a tensor $\mathcal{X} \in \mathbb{R}^{N_1 \times \cdots \times N_M}$ and a matrix $\mathbf{U} \in \mathbb{R}^{N_i \times N'}$ is the tensor in $\mathbb{R}^{N_1 \times \cdots \times N_{i-1} \times N' \times N_{i+1} \times \cdots \times N_M}$ defined by:*

$$\begin{aligned}&(\mathcal{X} \times_d \mathbf{U})(i_1, \ldots, i_{d-1}, j, i_{d+1}, \ldots, i_M) \\ &= \textstyle\sum_{i_d=1}^{N_i} \mathcal{X}(i_1, \ldots, i_{d-1}, i_d, i_{d+1}, \ldots, i_M)\mathbf{U}(i_d, j)\end{aligned} \quad (1)$$

for all index values.

Figure 4 shows an example of 3rd order tensor $\mathcal{X}$ mode-1 multiplies a matrix $\mathbf{U}$. The process is equivalent to first matricize $\mathcal{X}$ along mode-1, then to do matrix multiplication of $\mathbf{X}_1$ and $\mathbf{U}$, finally to fold the result back as a tensor.

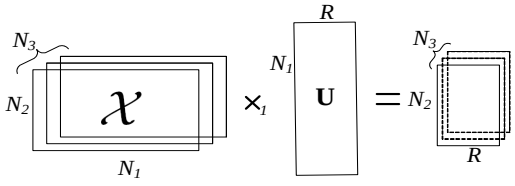


Figure 4: 3rd order tensor $\mathcal{X}_{[N_1,N_2,N_3]} \times_1 \mathbf{U}$ results in a new tensor in $\mathbb{R}^{R \times N_2 \times N_3}$

In general, a tensor $\mathcal{X} \in \mathbb{R}^{N_1 \times \cdots \times N_M}$ can multiply a sequence of matrices $\mathbf{U}_i|_{i=1}^M \in \mathbb{R}^{N_i \times R_i}$ as: $\mathcal{X} \times_1 \mathbf{U}_1 \cdots \times_M \mathbf{U}_M \in \mathbb{R}^{R_1 \times \cdots \times R_M}$,which can be written as $\mathcal{X} \prod_{i=1}^{M} \times_i \mathbf{U}_i$ for clarity. Furthermore, the notation for $\mathcal{X} \times_1 \mathbf{U}_1 \cdots \times_{i-1} \mathbf{U}_{i-1} \times_{i+1} \mathbf{U}_{i+1} \cdots \times_M \mathbf{U}_M$ (i.e. multiplication of all $\mathbf{U}_j$s except the i-th) is simplified as $\mathcal{X} \prod_{j \ne i} \times_j \mathbf{U}_j$.

DEFINITION 3 (RANK-$(R_1, \ldots, R_M)$ APPROXIMATION). *Given a tensor $\mathcal{X} \in \mathbb{R}^{N_1 \times \cdots \times N_M}$, a tensor $\tilde{\mathcal{X}} \in \mathbb{R}^{N_1 \times \cdots \times N_M}$ with rank $\left(\tilde{\mathbf{X}}_{(d)}\right) = R_d$ for $1 \le d \le M$, that minimizes the least-squares cost $\left\|\mathcal{X} - \tilde{\mathcal{X}}\right\|_F^2$, is the best rank-$(R_1, \cdots, R_M)$ approximation of $\mathcal{X}$.*[2]

The best rank approximation $\tilde{\mathcal{X}} = \mathcal{Y} \prod_{l=1}^{M} \times_l \mathbf{U}_l^T$, where the core tensor $\mathcal{Y} \in \mathbb{R}^{R_1 \times \cdots \times R_M}$ and the projection matrices $\mathbf{U}_l|_{l=1}^M \in \mathbb{R}^{N_l \times R_l}$.

3. PROBLEM DEFINITION

In this section, we first formally define the notations of *tensor sequence* and *tensor stream*. Then we overview the operations on them.

DEFINITION 4 (TENSOR SEQUENCE). *A sequence of Mth order tensor $\mathcal{X}_1 \ldots \mathcal{X}_n$, where each $\mathcal{X}_i \in \mathbb{R}^{N_1 \times \cdots \times N_M}$ $(1 \le i \le n)$, is called tensor sequence if n is a fixed natural number. And n is the cardinality of the tensor sequence.*

In fact, we can view an Mth order tensor sequence $\mathcal{X}_1 \ldots \mathcal{X}_n$ as a single $(M+1)$th order tensor with the dimensionality n on the additional mode.

DEFINITION 5 (TENSOR STREAM). *A sequence of Mth order tensor $\mathcal{X}_1 \ldots \mathcal{X}_n$, where each $\mathcal{X}_i \in \mathbb{R}^{N_1 \times \cdots \times N_M}$ $(1 \le i \le n)$, is called a tensor stream if n is a natural number that increases with time.*

Intuitively, we can consider a tensor stream are coming incrementally over time. And $\mathcal{X}_n$ is the latest tensor in the stream. In network monitoring example, a new 3rd order tensor (as the one in Figure 1(a)) comes every hour.

After defined the data models, the main operation is to represent the original tensors in other basis such that underlying patterns are easily revealed.

DEFINITION 6 (TENSOR ANALYSIS). *Given a sequence of tensors $\mathcal{X}_1 \ldots \mathcal{X}_n$, where each $\mathcal{X}_i \in \mathbb{R}^{N_1 \times \cdots \times N_M}$ $(1 \le i \le n)$, find the orthogonal matrices $\mathbf{U}_i \in \mathbb{R}^{N_i \times R_i}|_{i=1}^M$, one for each mode, such that the reconstruction error e is minimized:* $e = \sum_{t=1}^{n} \left\| \mathcal{X}_t - \mathcal{X}_t \prod_{i=1}^{M} \times_i (\mathbf{U}_i \mathbf{U}_i^T) \right\|_F^2$

Note that $\mathcal{X}_t \prod_{i=1}^{M} \times_i (\mathbf{U}_i \mathbf{U}_i^T)$ is the approximation of $\mathcal{X}_t$ under the space spanned by $\mathbf{U}_i|_{i=1}^M$. And it can be rewritten as

[2]The square Frobenius norm is defined as $\|\mathcal{X}\|_F^2 = \sum_{i=1}^{N_1} \cdots \sum_{i=1}^{N_M} \mathcal{X}(i_1, ..., i_M)^2$.

$\mathcal{Y}_t \prod_{i=1}^{M} \times_i \mathbf{U}_i^T$ where $\mathcal{Y}_t$ is the core tensor defined as $\mathcal{Y}_t = \mathcal{X}_t \prod_{i=1}^{M} \times_i \mathbf{U}_i$.

More specifically, we propose three variants under *tensor analysis*: offline tensor analysis (OTA) for a tensor sequence (see Section 4.1); and dynamic tensor analysis (DTA) and streaming tensor analysis (STA) for a tensor stream (see Section 4.2 and Section 4.3).

4. TENSOR ANALYSIS

First, Section 4.1 introduces *offline tensor analysis* (OTA) for a tensor sequence. Second, we present the core component, *dynamic tensor analysis* (DTA) (Section 4.2) and its variants. Third, Section 4.3 introduces *streaming tensor analysis* (STA), which approximates DTA to further reduce the time complexity.

4.1 Offline Tensor Analysis

We now introduce the *offline tensor analysis* (OTA) which is a generalization of PCA for higher order tensors[3].

Unlike PCA, which requires the input to be vectors (1st-order tensors), OTA can accept general Mth order tensors for dimensionality reduction. The formal definition is exactly the same as Definition 6. Figure 5 shows an example of OTA over n 2nd order tensors.

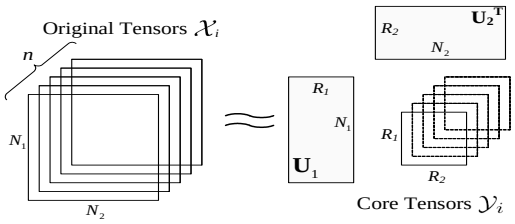


Figure 5: OTA projects n large 2nd order tensors $\mathcal{X}_i$ into n smaller 2nd order tensors $\mathcal{Y}_i$ with two projection matrices $\mathbf{U}_1$ and $\mathbf{U}_2$ (one for each mode).

Unfortunately, the closed form solution for OTA does not exist. An alternating projection is used to approach to the optimal projection matrices $U_l|_{l=1}^M$. The iterative algorithm converges in finite number of steps because each sub-optimization problem is convex. The detailed algorithm is given in Figure 6. Intuitively, it projects and matricizes along each mode; and it perform PCA to find the projection matrix for that mode. This process potentially needs performing more than one iteration.

In practice, this algorithm converges in a small number of iterations. In fact, Ding and Ye [7] shows the near-optimality of a similar algorithm for 2nd order tensors. Therefore, for time-critical applications, single iteration is usually sufficient to obtain the near-optimal projection matrices $\mathbf{U}_l|_{l=1}^M$. In that case, the algorithm essentially unfolds the tensor along each mode and applies PCA on those unfolded matrices separately.

Note that OTA requires all tensors available up front, which is not possible for dynamic environments (i.e., new tensors keep coming). Even if we want to apply OTA every time that a new tensor arrives, it is prohibitively expensive

[3]Similar ideas have been proposed for 2nd- or 3rd order tensors [21, 29, 12].

Input:
Training tensors $\mathcal{X}_i|_{i=1}^n \in \mathbb{R}^{N_1 \times \ldots \times N_M}$
The dimensionality of the output tensors $\mathcal{Y}_i \in \mathbb{R}^{R_1 \times \ldots \times R_M}$.
Output:
The projection matrix $\mathbf{U}_l|_{l=1}^M \in \mathbb{R}^{N_l \times R_l}$ constrained by $\|\mathbf{U}_l\| = \mathbf{I}$ and the output tensors $\mathcal{Y}_i|_{i=1}^n \in \mathbb{R}^{R_1 \times \ldots \times R_M}$.
Algorithm:
1. Set $\mathbf{U}_l$ to be $N_l \times R_l$ truncated identity matrix.
2. Conduct 3 - 8 iteratively.
3. For $d = 1$ to M
4. For $1 \le i \le n$
5. construct $\mathcal{X}_i^d = \mathcal{X}_i(\prod_{l \ne d} \times_l \mathbf{U}_l \mathbf{U}_l^T) \in \mathbb{R}^{R_1 \times \cdots \times N_d \times \cdots \times R_M}$
6. unfold($\mathcal{X}_i^d$, d) as $\mathbf{X}_{i(d)} \in \mathbb{R}^{N_d \times (\prod_{k \ne d} R_k)}$
7. construct variance matrix $\mathbf{C}_d = \sum_{i=1}^n \mathbf{X}_{i(d)}^T \mathbf{X}_{i(d)}$
8. compute $\mathbf{U}_d$ by diagonalizing $\mathbf{C}_d$
9. Check convergence: $Tr(||\mathbf{U}_l^T \mathbf{U}_l| - \mathbf{I}|) \le \varepsilon$ for $1 \le l \le M$.
10. Calculate the core tensor $\mathcal{Y}_i = \mathcal{X}_i \prod_{d=1}^{M} \times \mathbf{U}_d$ for $1 \le i \le n$

Figure 6: Algorithm: Offline Tensor Analysis

or merely impossible since the computation and space requirement are unbounded. Next we present an algorithm for the dynamic setting.

4.2 Dynamic Tensor Analysis

Here we present the dynamic tensor analysis (DTA), an incremental algorithm for tensor dimensionality reduction.
Intuition: The idea of incremental algorithm is to exploit two facts:

1. In general OTA can be computed relatively quickly once the variance matrices [4]are available;
2. Variance matrices can be incrementally updated without storing any historical tensor.

The algorithm processes each mode of the tensor at a time. In particular, the variance matrix of the dth mode is updated as:

$$\mathbf{C}_d \leftarrow \mathbf{C}_d + \mathbf{X}_{(d)}^T \mathbf{X}_{(d)}$$

where $\mathbf{X}_{(d)} \in \mathbb{R}^{(\prod_{i \ne d} N_i) \times N_d}$ is the mode-d matricizing of the tensor $\mathcal{X}$. The updated projection matrices can be computed by diagonalization: $\mathbf{C}_d = \mathbf{U}_d \mathbf{S}_d \mathbf{U}_d^T$, where $\mathbf{U}$ is orthogonal matrix and $\mathbf{S}$ is diagonal matrix. The pseudo-code is listed in Figure 7. The process is also visualized in Figure 8.
Forgetting factor: Dealing with time dependent models, we usually do not treat all the timestamps equally. Often the recent timestamps are more important than the ones far away from the past [5]. More specifically, we introduce a forgetting factor into the model. In terms of implementation, the only change to the algorithm in Figure 7:

$$\mathbf{C}_d \leftarrow \lambda \mathbf{C}_d + \mathbf{X}_{(d)}^T \mathbf{X}_{(d)}$$

[4]Recall the variance matrix of $\mathbf{X}_{(d)} \in \mathbb{R}^{(\prod_{i \ne d} N_i) \times N_d}$ is defined as $\mathbf{C} = \mathbf{X}_{(d)}^T \mathbf{X}_{(d)} \in \mathbb{R}^{N_d \times N_d}$.
[5]Unless there is a seasonality in the data, which is not discussed in the paper.

Input:
New tensor $\mathcal{X} \in \mathbb{R}^{N_1 \times \ldots \times N_M}$
Previous projection matrices $\mathbf{U}_i|_{i=1}^M \in \mathbb{R}^{N_i \times R_i}$
Previous energy matrices $\mathbf{S}_i|_{i=1}^M \in \mathbb{R}^{R_i \times R_i}$
Output:
New projection matrices $\mathbf{U}_i|_{i=1}^M \in \mathbb{R}^{N_i \times R'_i}$
New energy matrices $\mathbf{S}_i|_{i=1}^M \in \mathbb{R}^{R'_i \times R'_i}$
Core tensor $\mathcal{Y} \in \mathbb{R}^{R'_1 \times \cdots \times R_M}$
Algorithm:
// update every mode
1. For $d = 1$ to M
2. Mode-d matricize $\mathcal{X}$ as $\mathbf{X}_{(d)} \in \mathbb{R}^{(\prod_{i \neq d} N_i) \times N_d}$
3. Reconstruct variance matrix $\mathbf{C}_d \leftarrow \mathbf{U}_d \mathbf{S}_d \mathbf{U}_d^T$
4. Update variance matrix $\mathbf{C}_d \leftarrow \mathbf{C}_d + \mathbf{X}_{(d)}^T \mathbf{X}_{(d)}$
5. Diagonalization $\mathbf{C}_d = \mathbf{U}_d \mathbf{S}_d \mathbf{U}_d^T$
6. Compute new rank R_d and truncate $\mathbf{U}$ and $\mathbf{S}$
7. Calculate the core tensor $\mathcal{Y} = \mathcal{X} \prod_{d=1}^{M} \times \mathbf{U}_d$

Figure 7: Algorithm: Dynamic Tensor Analysis

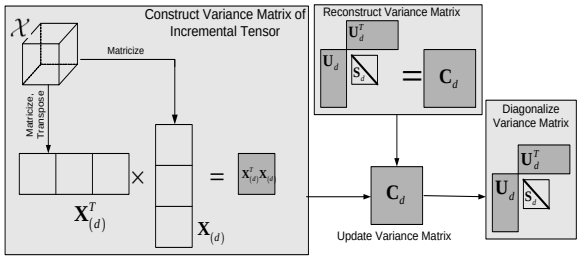


Figure 8: New tensor $\mathcal{X}$ is matricized along the dth mode. Then variance matrix $\mathbf{C}_d$ is updated by $\mathbf{X}_{(d)}^T \mathbf{X}_{(d)}$. The projection matrix $\mathbf{U}_d$ is computed by diagonalizing $\mathbf{C}_d$.

where λ is the forgetting factor between 0 and 1. Herein $\lambda = 0$ when no historical tensors are considered, while $\lambda = 1$ when historical tensors have the same weights as the new tensor $\mathcal{X}$. Note that forgetting factor is a well-known trick in signal processing and adaptive filtering [11].

Update Rank: One thing missing from Figure 7 is how to update the rank R_i. In general the rank can be different on each mode or can change over time. The idea is to keep the smallest rank R_i such that the energy ratio $\|\mathbf{S}_i\|_F / \|\mathbf{X}\|_F^2$, is above the threshold [15]. In all experiments, we set the threshold as 0.9 unless mentioned otherwise.

Complexity: The space consumption for incremental algorithm is $\prod_{i=1}^M N_i + \sum_{i=1}^M N_i \times R_i + \sum_{i=1}^M R_i$. The dominate factor is from the first term $\mathrm{O}(\prod_{i=1}^M N_i)$. However, standard OTA requires $\mathrm{O}(n \prod_{i=1}^M N_i)$ for storing all tensors up to time n, which is unbounded.

The computation cost is $\sum_{i=1}^M R_i N_i^2 + n \prod_{i=1}^M N_i + \sum_{i=1}^M R'_i N_i^2$. Note that for medium or low mode tensor (i.e., order $M \leq 5$), the diagonalization is the main cost. Section 4.3 introduces a faster approximation of DTA that avoids diagonalization.

While for high order tensors (i.e., order $M > 5$), the dominate cost becomes $\mathrm{O}(\prod_{i=1}^M N_i)$ from updating the variance matrix (line 4). Nevertheless, the improvement of DTA is still tremendous compared to OTA $\mathrm{O}(n \prod_{i=1}^M N_i)$ where n is the number of all tensors up to current time.

4.3 Streaming Tensor Analysis

In this section, we present the *streaming tensor analysis* (STA), a fast algorithm to approximate DTA without diagonalization. We first introduce the key component of tracking a projection matrix. Then a complete algorithm is presented for STA.

Tracking a projection matrix: For most of time-critical applications, the diagonalization process (in Figure 7) for every new tensor can be expensive. Especially, when the change of the variance matrix is small, it is not worthy diagonalizing that matrix. The idea is to continuously track the changes of projection matrices using the deflation technique for sequentially estimating the principal components [24].

The main idea behind the tracking algorithm (see Figure 9) is to read in a new vector (one row of a tensor matrix) and perform three steps:

1. Compute the projection $\mathbf{y}$, based on the *current* projection matrix $\mathbf{U}$, by projecting $\mathbf{x}$ onto $\mathbf{U}$;
2. Estimate the reconstruction error ($\mathbf{e}$) and the energy, based on the $\mathbf{y}$ values; and
3. Update the estimates of $\mathbf{U}$.

Intuitively, the goal is to adaptively update $\mathbf{U}$ quickly based on the new tensor. The larger the error $\mathbf{e}$, the more $\mathbf{U}$ is updated. However, the magnitude of this update should also take into account the past data currently "captured" by $\mathbf{U}$. For this reason, the update is inversely proportional to the current *energy* $\mathbf{s}(i)$.

Input:
projection matrix $\mathbf{U} \in \mathbb{R}^{n \times r}$
energy vector $\mathbf{s} \in \mathbb{R}^n$
input vector $\mathbf{x} \in \mathbb{R}^n$
Output:
updated projection matrix $\mathbf{U}$
updated energy vector $\mathbf{s}$
1. As each point $\mathbf{x}_{t+1}$ arrives, initialize $\acute{\mathbf{x}} := \mathbf{x}_{t+1}$.
2. For $1 \leq i \leq r$

$$\mathbf{y}(i) = \mathbf{U}(:,i)^T \acute{\mathbf{x}}_i \quad \text{(projection onto } \mathbf{U}(:,i))$$
$$\mathbf{s}(i) \leftarrow \mathbf{s}(i) + \mathbf{y}(i)^2 \quad \text{(energy } \propto i\text{-th eigenval)}$$
$$\mathbf{e} = \acute{\mathbf{x}} - \mathbf{y}(i)\mathbf{U}(:,i) \quad \text{(error, } \mathbf{e} \perp \mathbf{U}(:,i))$$
$$\mathbf{U}(:,i) \leftarrow \mathbf{U}(:,i) + \frac{1}{\mathbf{s}(i)} \mathbf{y}(i)\mathbf{e} \quad \text{(update PC estimate)}$$
$$\acute{\mathbf{x}}(i+1) \leftarrow \acute{\mathbf{x}}_i - \mathbf{y}(i)\mathbf{U}(:,i) \quad \text{(repeat with remainder).}$$

Figure 9: Tracking a projection matrix

Streaming tensor analysis: The goal of STA is to adjust projection matrices smoothly as the new tensor comes in. Note that the tracking process has to be run on all modes of the new tensor. For a given mode, we first matricize the tensor $\mathcal{X}$ into a matrix $\mathbf{X}_{(d)}$ (line 2 of Figure 10), then adjust the projection matrix $\mathbf{U}_d$ by applying *tracking a projection matrix* over the rows of $\mathbf{X}_{(d)}$. The full algorithm is in Figure 10. And the process is visualized in Figure 11.

To further reduce the time complexity, we can select only a subset of the vectors in $\mathbf{X}_{(d)}$. For example, we can sample vectors with high norms, because those potentially give higher impact to the projection matrix.

Complexity: The space complexity of STA is ITPCA, which is only the size of the new tensor. The computational complexity is $O((\sum_i R_i)\prod_i N_i)$ which is smaller than ITPCA (when $R_i \ll N_i$). The STA can be further improved with random sampling technique, i.e., use only subset of rows of $\mathbf{X}_{(d)}$ for update.

Input:
New tensor $\mathcal{X} \in \mathbb{R}^{N_1 \times \ldots \times N_M}$
Old projection matrices $\mathbf{U}_i|_{i=1}^M \in \mathbb{R}^{N_i \times R_i}$
Old energy matrices $\mathbf{S}_i|_{i=1}^M \in \mathbb{R}^{R_i \times R_i}$
Output:
New projection matrices $\mathbf{U}_i|_{i=1}^M \in \mathbb{R}^{N_i \times R_i}$
New energy matrices $\mathbf{S}_i|_{i=1}^M \in \mathbb{R}^{R_i \times R_i}$
Core tensor $\mathcal{Y} \in \mathbb{R}^{R_1 \times \cdots \times R_M}$
Algorithm:

1. For $d = 1$ to M // update every mode
2. Matricize new tensor $\mathcal{X}$ as $\mathbf{X}_{(d)} \in \mathbb{R}^{(\prod_{i \neq d} N_i) \times N_d}$
3. For each column vector $\mathbf{x}$ in $\mathbf{X}_{(d)}^T$
4. Track projection matrix $\mathbf{U}_d$, $\text{diag}(\mathbf{S}_i)$, $\mathbf{x}$
5. Calculate the core tensor $\mathcal{Y} = \mathcal{X} \prod_{d=1}^{M} \times \mathbf{U}_d$

Figure 10: Algorithm: Streaming Tensor Analysis

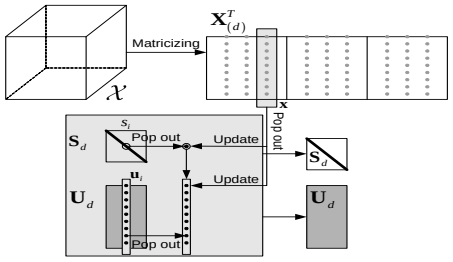


Figure 11: New tensor $\mathcal{X}$ is matricized along the dth mode. For every row of $\mathbf{X}_d$, we update the projection matrix $\mathbf{U}_d$. And $\mathbf{S}_d$ helps determine the update size.

5. APPLICATIONS

After presenting the core technique of tensor analysis, we now illustrate two practical applications of DTA or STA: 1) *Anomaly detection*, which tries to identify abnormal behavior across different tensors as well as within a tensor; 2) *Multi-way latent semantic indexing (LSI)*, which finds the correlated dimensions in the same mode and across different modes.

5.1 Anomaly Detection

We envision the abnormal detection as a multi-level screening process, where we try to find the anomaly from the broadest level and gradually narrow down to the specifics. In particular, it can be considered as a three-level process for tensor stream: 1) given a sequence of tensors, identify the abnormal ones; 2) on those suspicious tensors, we locate the abnormal modes; 3) and then find the abnormal dimensions of the given mode. In the network monitoring example, the system first tries to determine when the anomaly occurs; then it tries to find why it occurs by looking at the traffic patterns from sources, destinations and ports, respectively; finally, it narrows down the problem on specific hosts or ports.

For level one (tensor level), we model the abnormality of a tensor $\mathcal{X}$ by its reconstruction error:

$$e_i = \|\mathcal{X}_i - \mathcal{Y}_i \prod_{l=1}^{M} \times_l \mathbf{U}_l^T\|_F = \|\mathcal{X}_i - \mathcal{X}_i \prod_{l=1}^{M} \times_l \mathbf{U}_l \mathbf{U}_l^T\|_F^2.$$

For level two (mode level), the reconstruction error of the lth mode only involves one projection matrix $\mathbf{U}_d$ for a given tensor $\mathcal{X}$:

$$e_d = \|\mathcal{X} - \mathcal{X} \times_l \mathbf{U}_l \mathbf{U}_l^T\|_F^2.$$

For level three (dimension level), the error of dimension d on the lth mode is just the reconstruction of the tensor slice of dimension d along the lth mode.

How much error is too much? We answer this question in the typical way: If the error at time T is enough (say, 2) standard deviations away from the mean error so far, we declare the tensor $\mathcal{X}_T$ as abnormal. Formally, the condition is as follows:

$$e_{T+1} \geq \text{mean}\left(e_i|_{i=1}^{T+1}\right) + \alpha \cdot \text{std}\left(e_i|_{i=1}^{T+1}\right).$$

5.2 Multi-way Latent Semantic Indexing

The goal of the multi-way LSI is to find high correlated dimensions within the same mode and across different modes, and monitor them over time. Consider the DBLP example, author-keyword over time, Figure 12 shows that initially (in $\mathcal{X}_1$) there is only one group, DB, in which all authors and keywords are related to databases; later on (in $\mathcal{X}_n$) two groups appear, namely, databases (DB) and data mining (DM).

Correlation within a mode: A projection matrix gives the correlation information among dimensions for a given mode. More specifically, the dimensions of the lth mode can be grouped based on their values in the columns of $\mathbf{U}_l$. The entries with high absolute values in a column of $\mathbf{U}_l$ correspond to the important dimensions in the same "concept".

In the DBLP example showing in Figure 12, $\mathbf{U}_K$ corresponds to the keyword concepts. First and second columns are the DB and DM concepts, respectively. The circles of $\mathbf{U}_A$ and $\mathbf{U}_K$ indicate the influential authors and keywords in DM concept, respectively. Similarly, the stars are for DB concept. An example of actual keywords and authors is in Table 3.

Correlations across modes: The interesting aspect of DTA is that the core tensor $\mathcal{Y}$ provides indications on the correlations of different dimensions across different modes. More specifically, a large entry in $\mathcal{Y}$ means high correlation between the corresponding columns in all modes. For example in Figure 12, the large values of $\mathcal{Y}_i$ (in the shaded region) activate the corresponding concepts of the tensor $\mathcal{X}_i$. For simplicity, we described a non-overlapping example, however, groups may overlap which actually happens often in real datasets.

Correlations across time: And the core tensor $\mathcal{Y}_i$s also capture the temporal evolution of concepts. In particular, $\mathcal{Y}_1$ only activates the DB concept; while $\mathcal{Y}_n$ activates both DB and DM concepts.

Authors	Keywords	Year
michael carey,michael stonebraker, h. jagadish,hector garcia-molina	queri,parallel,optimization,concurr,objectorient	1995
surajit chaudhuri,mitch cherniack,michael stonebraker,ugur etintemel	distribut,systems,view,storag,servic,process,cach	2004
jiawei han,jian pei,philip s. yu,jianyong wang,charu c. aggarwal	streams,pattern,support,cluster,index,gener,queri	2004

Table 3: Example clusters: first two lines databases groups, last line data mining group.

Note that DTA monitors the projection matrices over time. In this case, the concept space captured by projection matrix $\mathbf{U}_i$s are also changing over time.

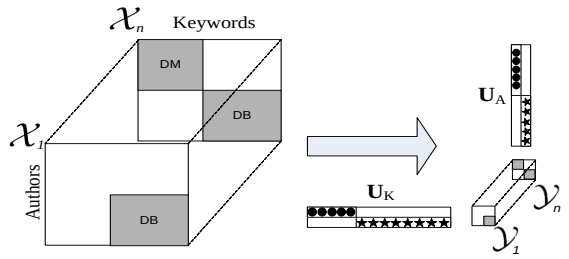


Figure 12: $\mathbf{U}_A$ and $\mathbf{U}_K$ capture the DB (stars) and DM (circles) concepts in authors and keywords, respectively; initially, only DB is activated in $\mathcal{Y}_1$; later on both DB and DM are in $\mathcal{Y}_n$.

6. EXPERIMENTS

In this Section, we evaluate the proposed methods on real datasets. Section 6.1 introduces the datasets. Section 6.2 studies the computation efficiency of different methods.

6.1 Datasets

name	description	dimension	timestamps
IP2D	Network 2D	500-by-500	1200
IP3D	Network 3D	500-by-500-by-100	1200
DBLP	DBLP data	4584-by-3741	11

Figure 13: Three datasets

The Network Datasets: IP2D, IP3D

The traffic trace consists of TCP flow records collected at the backbone router of a class-B university network. Each record in the trace corresponds to a directional TCP flow between two hosts through a server port with timestamps indicating when the flow started and finished.

With this traffic trace, we study how the communication patterns between hosts and ports evolve over time, by reading traffic records from the trace, simulating network flows arriving in real time. We use a window size of Δt seconds to construct a source-destination 2nd order tensors and source-destination-port 3rd order tensor every hour ($\Delta t = 3600$ seconds). For each 2nd order tensor, the modes correspond to source and destination IP addresses, respectively, with the value of each entry (i, j) representing the total number of TCP flows (packets) sent from the i-th source to the j-th destination during the corresponding Δt seconds. Similarly, the modes for each 3rd order tensor corresponds to the source-IP address, the destination-IP address and the server port number, respectively.

Because the tensors are very sparse and the traffic flows are skewed towards a small number of dimensions on each mode, we select only N_1=N_2=500 sources and destinations and N_3=100 port numbers with high traffic. Moreover, since the values are very skewed, we scaled them by taking the logarithm (and actually, $\log(x + 1)$, to account for $x = 0$), so that our tensor analysis is not dominated by a few very large entries. All figures are constructed over a time interval of 1,200 timestamps(hours).

DBLP Bibliographic Data Set:

Based on DBLP data [1], we generate author-keyword 2nd order tensors of KDD and VLDB conferences from year 1994 to 2004 (one tensor per year). The entry (a, k) in such a tensor indicates that author a has published a paper with keyword k in the title during that year. The value of the entry (a, k) is the number of times k appear in the title during that year. In total, there are 4,584 authors and 3,741 keywords. Note that the keywords are generated from the paper title after proper stemming and stop-word removal.

All the experiments are performed on the same dedicated server with four 2.4GHz Xeon CPUs and 12GB memory. For each experiment, we repeat it 10 times, and report the mean.

6.2 Efficiency Evaluation

Computational cost:

We first compare three different methods, namely, offline tensor analysis (OTA), dynamic tensor analysis (DTA), streaming tensor analysis (STA), in terms of computation time for different datasets. Figure 14 shows the CPU time in logarithm as a function of relapse time. Since the new tensors keep coming, the cost of OTA increases linearly[6]; while DTA and STA remains more or less constant. Note that DBLP in Figure 14(c) shows lightly increasing trend on DTA and STA because the tensors become denser over time (i.e., the number of published paper per year are increasing over time), which affects the computation cost slightly.

We show that STA provides an efficient way to approximate DTA over time, especially with sampling. More specifically, after matricizing, we sample the vectors with high norms to update the projection matrices. Figure 15 shows the CPU time vs. sampling rate, where STA runs much faster compared to DTA.

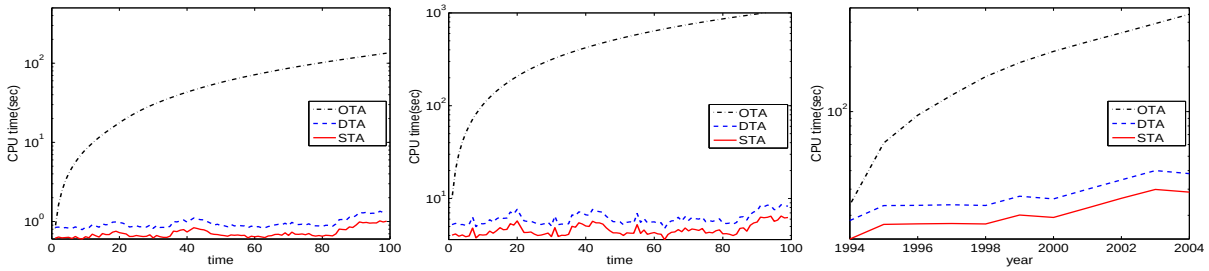
Accuracy comparison:

Now we evaluate the approximation accuracy of DTA and STA compared to OTA.

Performance metric: Intuitively, the goal is to be able to compare how accurate each tensor decomposition is to the original tensors. Therefore, reconstruction error provides a natural way to quantify the accuracy. Recall the reconstruction error is defined in Definition 6. Error can always be reduced when more eigenvectors are included (more columns in the projection matrices). Therefore, we fix the number of eigenvectors in the projection matrices for all three methods such that the reconstruction error for OTA is 20%. And we use the error ratios between DTA/STA to OTA as the performance indices.

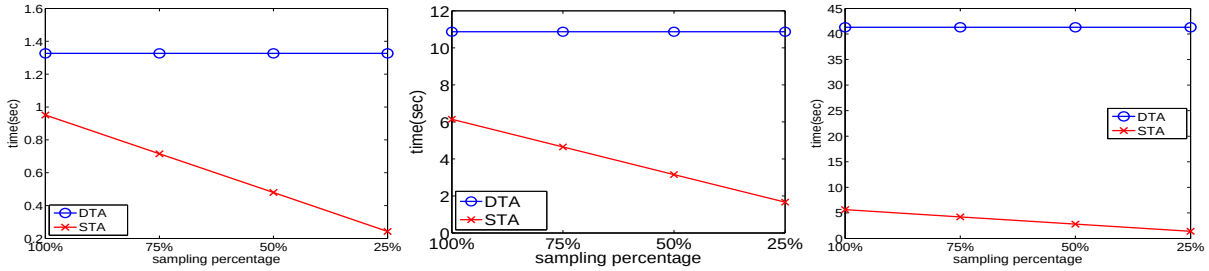
Evaluation results: Overall the reconstruction error of

[6]We estimate CPU time by extrapolation because OTA runs out of the memory after a few timestamps.



(a)IP2D (b) IP3D (c) DBLP

Figure 14: Both DTA and STA use much less time than OTA over time across different datasets



(a) IP2D (b) IP3D (c) DBLP

Figure 15: STA uses much less CPU time than DTA across different datasets

DTA and STA are close to the expensive OTA method (see Figure 16(d)). Note that the cost for doing OTA is very expensive in both space and time complexity. That is why only a few timestamps are shown in Figure 16 since after that point OTA runs out of the memory.

In more details, Figure 16(a)-(c) plot the error ratios over time for three datasets. There we also plot the one that never updates the original projection matrices as a lower-bound baseline.

DTA performs very close to OTA, which suggests a cheap incremental methods over the expensive OTA. And an even cheaper method, STA, usually gives good approximation to DTA (see Figure 16(a) and (b) for IP2D and IP3D). But note that STA performs considerably worse in DBLP in Figure 16(c) because the adaptive subspace tracking technique as STA cannot keep up to the big changes of DBLP tensors over consecutive timestamps. Therefore, STA is only recommended for the fast incoming with significant time-dependency (i.e., the changes over consecutive timestamps should not be too big).

7. DATA MINING CASE STUDIES

After we evaluated the efficiency of the proposed methods, we now present two data mining applications using both our proposed methods, the *dynamic* and the *streaming tensor analysis*. The two applications are 1) *anomaly detection* for network traffic analysis and *multi-way LSI* on DBLP data.

7.1 Anomaly Detection

Here we focus on the `IP3D` dataset, that is, source-destination-port tensors, and we use the algorithm described in Section 5.1 to detect the following three types of anomalies:

Abnormal source hosts: For example, port-scanners that send traffic to a large number of different hosts in the system.

Abnormal destination hosts: For example, the victim of a distributed denial of service attack (DDoS), which receives high volume of traffic from a large number of source hosts.

Abnormal ports: Ports that receive abnormal volume of traffic, and/or traffic from too many hosts.

Experiment Setup: We randomly pick one tensor from normal periods with no known attacks. Due to the lack of detailed anomaly information, we manually inject anomalies into the selected timestamps using the following method. For a given mode (i.e., source, destination or port) we randomly select a dimension and then set 50% of the corresponding slice to 1 simulating an attack in the network.

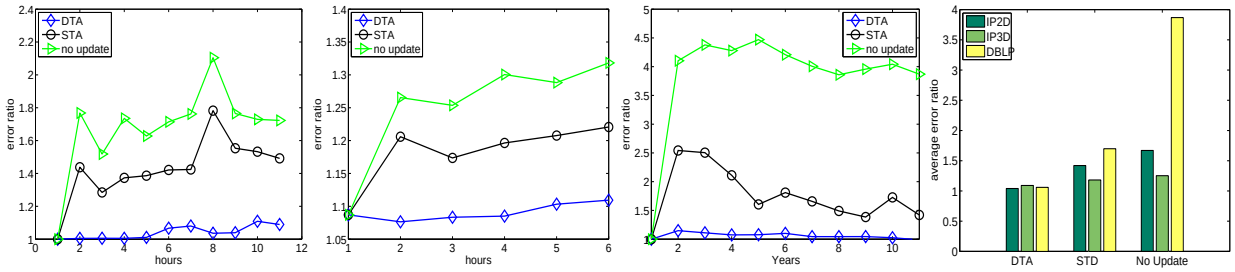
There are one additional input parameter: forgetting factor α which is varied from .2 to 1. The result is obtained using DTA, and STA achieved very similar result so we ignore it for clarity.

Performance Metrics: We use detection precision as our metric. We sort dimensions based their reconstruction error, and extract the smallest number of top ranked hosts (say k hosts) that we need to select as suspicious hosts, in order to detect all injected abnormal host (i.e., recall = 100% with no false negatives). Precision thus equals $1/k$, and the false positive rate equals $1 - \text{precision}$. We inject only one abnormal host each time. And we repeat each experiment 100 times and take the mean.

Results: Table 4 shows the precision vs. forgetting factor for detecting three different anomalies. Although the precision remains high for both types of anomaly detection, we achieve a higher precision when forgetting factor is low

Ratio	20%	40%	60%	80%	100%
Source IP	1	0.99	0.99	0.97	0.95
Destination IP	1	0.99	0.99	0.96	0.94
Port	1	1	0.99	0.98	0.97

Table 4: Network anomaly detection: precision is very high (the detection false positive rate = 1 − precision).



(a) IP2D (b) IP3D (c) DBLP (d) Average

Figure 16: Reconstruction error over time

meaning it forgets faster. This is because the attacks we injected take place immediately which do not depend on the past; in this case, fast forgetting is good. However, for more elaborate attacks, the forgetting factor may need to be set differently.

Identify real anomaly: For network traffic, normal host communication patterns in a network should roughly be similar to each other over time. A sudden change of approximation accuracy suggests structural changes of communication patterns since the same approximation procedure can no longer keep track of the overall patterns. Figure 17(a) shows relative reconstruction error[7] over time using DTA. The anomaly points are the ones above the red line, which is the threshold based on 3 standard deviation above the mean error percentage. The overall accuracy remains high. But a few unusual error bursts occurs at hour 140 and 160 (circle in Figure 17(a). We manually investigate into the trace further, and indeed find the onset of worm-like hierarchical scanning activities. Figure 17 (b) and (c) shows the normal (green dash circle in (a)) and abnormal (red solid circle in (a)) communication patterns. The dot in Figure 17 means there are packet flows between the corresponding source and destination. The prominent difference between these two is that there are more scanning sources (more columns with many dots). We can successfully identify this real anomaly in the data using reconstruction error from DTA.

7.2 Multi-way LSI

After showing the use of reconstruction error for anomaly detection, we now illustrate another important application of DTA/STA, i.e., *multi-way LSI*. The idea is presented in Section 5.2. Due to the lack of ground truth, we provide manual inspection on the clusters. And we now present some of observation and examples.

Clustering network data: We perform 3-way LSI on Network 3D data. The cluster is a tuple with three sets for source, destination and port, respectively. Several interesting clusters are verified by domain experts. For example, the algorithm successfully group web server ports into one cluster while mail server ports into another. And the destination hosts in those clusters are confirmed to be web server and mail servers.

Clustering DBLP data: Similarly, we perform 2-way LSI on DBLP datasets over time. Two example clusters are listed in Table 3. The algorithm correctly separate the two groups (data mining, databases) of people and concepts (2nd and 3rd clusters). And it identifies the focus change over time as well, e.g., 1st and 2nd groups are both about databases, but the focus has changed from object-oriented (1995) to stream (2004).

[7]Relative error is the reconstruction error by the input tensor norm.

8. RELATED WORK

Tensor and Multilinear Analysis: Tensor algebra and multilinear analysis have been applied successfully in many domains [5, 17, 28]. Powerful tools have been proposed, including Tucker decomposition [26], parallel factor analysis [10] or canonical decomposition [4], and bilinear PCA. Tensors have been recently used in machine vision research, for example by Shashua and Levin [25] for linear image coding, by Vasilescu and Terzopoulos [27] for face recognition. Ye [30] presented the generalized low rank approximations which extends PCA from the vectors (1st-order tensors) into matrices (2nd order tensors). Ding and Ye [7] proposed an approximation of [30]. Similar approach is also proposed in [12]. Xu et al. [29] formally presented the tensor representation for PCA and applied it for face recognition. Drineas and Mahoney [8] showed how to approximate the tensor SVD using biased sampling.

These methods do one or more of the following assumptions: the dataset is dense, or static. We are interested in sparse, streams of tensors, like the IP traffic matrices over time.

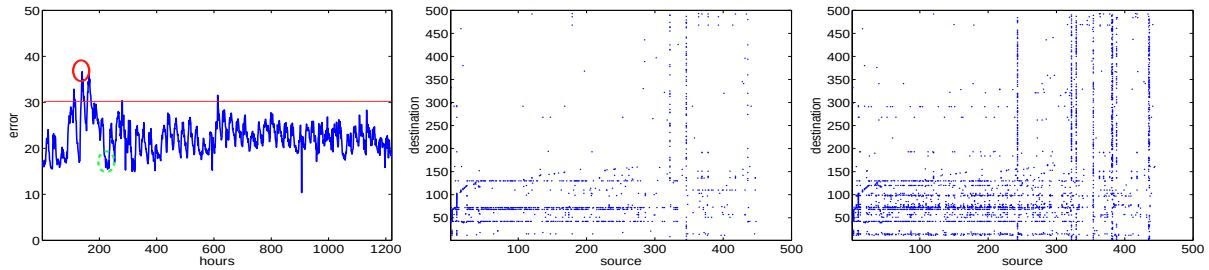
Stream and Graph Mining: Data streams has been extensively studied in recent years. A recent surveys [22] have discussed many data streams algorithms. Among them, Papadimitriou et al. [24] proposes a online algorithm that summarizes the multiple streams incrementally. Our STA algorithm has a similar flavor but for general tensors instead of just vectors.

A 2nd order tensor is a matrix and can be seen as a graph. Analysis and mining of static graphs has attracted a lot of interest, with many graph partitioning methods, including METIS [18], spectral partitioning [16], information-theoretic methods [6] and several variations. All these works focus on static graphs (2nd order tensors), while our emphasis is on time-evolving sequences of graphs and in general, sequences of tensors of potentially even higher order.

9. CONCLUSIONS

Numerous mining applications can be handled by matrices, the powerful SVD/PCA, and its variants and extensions. However, they all fall short when we want to study multiple modes, for example, time-evolving traffic matrices, time-evolving dataCubes, social networks with labeled edges, to name a few.

Next we show how to solve these higher order problems, by introducing the concept and vast machinery of *tensors*.



(a) error over time (b) normal (c) abnormal

Figure 17: Network flow over time: we can detect anomalies by monitoring the approximation accuracy

Our contributions are the following:

We introduce *tensors* and specifically *tensor stream* to solve even more general streaming problems than in the past. Our approach is applicable to all time-evolving settings, including as co-evolving time series, data streams and sensor networks, time-evolving graphs (even labeled ones), time-evolving social networks.

We propose two new tools, the *dynamic* and the *streaming tensor analysis* (DTA and STA) which incrementally mine and summarize large tensors, saving space and detecting patterns. DTA and STA are *fast*, *nimble* (since they avoid storing old tensors), and *fully automatic*, without requiring any user-defined parameters.

We provide experiments on two real, large datasets: network flow data and DBLP data. With respect to efficiency, our DTA and STA methods gives *several orders of magnitude* speed-up over the offline tensor analysis, with a small loss of accuracy. With respect to effectiveness, we applied our methods to anomaly detection and "multi-way LSI"; in both settings our tools found interesting and explainable patterns.

Future work includes the generalization into tensors for other matrix decomposition methods (like ICA, multinomial PCA); the integration of tensor analysis with time series forecasting (ARIMA for tensors, instead of scalars); the use of additional, static tensor decomposition methods like PARAFAC and high-order SVD [5], to name a few.

10. REFERENCES

[1] http://www.informatik.uni-trier.de/ ley/db/.

[2] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Mining association rules between sets of items in large databases. In *SIGMOD*, pages 207–216, 1993.

[3] S. Brin and L. Page. The anatomy of a large-scale hypertextual (web) search engine. In *WWW*, pages 107–117, 1998.

[4] J. D. Carroll and J. Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of 'eckart-young' decomposition. *Psychometrika*, 35(3):283–319, 1970.

[5] L. de Lathauwer. *Signal Processing Based on Multilinear Algebra*. PhD thesis, Katholieke, University of Leuven, Belgium, 1997.

[6] I. S. Dhillon, S. Mallela, and D. S. Modha. Information-theoretic co-clustering. In *KDD*, 2003.

[7] Chris Ding and Jieping Ye. Two-dimensional singular value decomposition (2dsvd) for 2d maps and images. In *SDM*, 2005.

[8] Petros Drineas and Michael W. Mahoney. A randomized algorithm for a tensor-based generalization of the svd. *technical report*.

[9] Peter W. Foltz and Susan T. Dumais. Personalized information delivery: An analysis of information filtering methods. *Comm. of ACM (CACM)*, 35(12), 1992.

[10] R.A. Harshman. Foundations of the parafac procedure: model and conditions for an explanatory multi-mode factor analysis. *UCLA working papers in phonetics*, 16:1–84, 1970.

[11] S. Haykin. *Adaptive Filter Theory*. Prentice Hall, 1992.

[12] Xiaofei He, Deng Cai, and Partha Niyogi. Tensor subspace analysis. In *NIPS*, 2005.

[13] Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *KDD*, 2001.

[14] Piotr Indyk, Nick Koudas, and S. Muthukrishnan. Identifying representative trends in massive time series data sets using sketches. In *VLDB*, pages 363–372, 2000.

[15] I.T. Jolliffe. *Principal Component Analysis*. Springer, 2002.

[16] R. Kannan, S. Vempala, and A. Vetta. On clusterings – good, bad and spectral. In *FOCS*, 2000.

[17] A. Kapteyn, H. Neudecker, and T. Wansbeek. An approach to n-mode component analysis. *Psychometrika*, 51(2):269–275, 1986.

[18] George Karypis and Vipin Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48(1):96–129, 1998.

[19] Jon Kleinberg. Authoritative sources in a hyperlinked environment. In *SODA*, 1998.

[20] Flip Korn, Alexandros Labrinidis, Yannis Kotidis, and Christos Faloutsos. Quantifiable data mining using ratio rules. *VLDB Journal*, 8(3-4):254–266, 2000.

[21] P. Kroonenberg and J. D. Leeuw. Principal component analysis of three-mode data by means of alternating least square algorithms. *Psychometrika*, 45:69–97, 1980.

[22] S. Muthukrishnan. *Data streams: algorithms and applications*, volume 1. Foundations and Trends. in Theoretical Computer Science, 2005.

[23] Christos H. Papadimitriou, Prabhakar Raghavan, Hisao Tamaki, and Santosh Vempala. Latent semantic indexing: A probabilistic analysis. In *PODS*, pages 159–168, 1998.

[24] Spiros Papadimitriou, Jimeng Sun, and Christos Faloutsos. Streaming pattern discovery in multiple time-series. In *VLDB*, pages 697–708, 2005.

[25] Amnon Shashua and Anat Levin. Linear image coding for regression and classification using the tensor-rank principle. In *CVPR*, number 1, pages 42–49, 2001.

[26] L. R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966.

[27] M. A. O. Vasilescu and D. Terzopoulos. Multilinear analysis of image ensembles: Tensorfaces. In *ECCV*, pages 447–460, 2002.

[28] N. Viereck, M. Dyrby, and S. B. Engelsen. *Monitoring Thermal Processes by NMR Technology*. Elsevier Academic Press, 2006.

[29] Dong Xu, Shuicheng Yan, Lei Zhang, Hong-Jiang Zhang, Zhengkai Liu, and Heung-Yeung Shum. Concurrent subspaces analysis. In *CVPR*, pages 203–208, 2005.

[30] Jieping Ye. Generalized low rank approximations of matrices. *Machine Learning*, 61:167–191, 2004.

[31] Lizhuang Zhao and Mohammed J. Zaki. Tricluster: An effective algorithm for mining coherent clusters in 3d microarray data. In *SIGMOD*, pages 694–705, 2005.