# Incremental Pattern Discovery on Streams, Graphs and Tensors

Jimeng Sun

## Abstract

Incremental pattern discovery targets at streaming applications where the data are arriving continuously in real-time. How to find patterns (main trends) in real-time? How to efficiently update the old patterns when new data arrive? How to utilize the pattern to solve other problem such as anomaly detection?

For example, 1) a sensor network monitors a large number of distributed streams (such as temperature and humidity); 2) network forensics monitor the Internet communication patterns to identify the attacks; 3) cluster monitoring examines the system behaviors of a number of machines for potential failures; 4) social network analysis monitors a dynamic graph for communities and abnormal individuals; 5) financial fraud detection tries to find fraudulent activities from a large number of transactions in real-time.

In this thesis proposal, we first investigate a powerful data model *tensor stream* (TS) where there is one tensor per timestamp. To capture diverse data formats: we have a zero-order TS for a single time-series (stock price for google over time), a first-order TS for multiple time-series (e.g., sensor measurement streams), a second-order TS for a matrix (e.g., graphs), and a high-order TS for a multi-array (e.g. Internet communication network, source-destination-port). Second, we develop different online algorithms on TS: 1) the centralized and distributed SPIRIT for mining a first-order TS [30, 33]; 2) compact matrix decomposition (CMD) for a second-order TS [35]; 3) the dynamic tensor analysis (DTA) and streaming tensor analysis (STA) for a high-order TS [34]. From the methodology aspect, we propose to extend CMD for incremental data and generalize the methods to handle more constraints (such as nonnegativity) and different distance measures (e.g. Bregman divergence). From the evaluation aspect, we propose to apply our methods in depth to some real applications such as network forensics, cluster monitoring and financial fraud detection.

## Contents

## 1 Introduction

Incremental pattern discovery targets at streaming applications where data are arriving continuously in real-time. The goal is to answer the following questions: How to find patterns (main trends) in real-time? How to efficiently update the old patterns when new data arrive? How to utilize the pattern to solve other problem such as anomaly detection and clustering?

Some examples include:

- *Sensor Networks* monitor different measurements (such as temperature and humidity) from a large number of distributed sensors. The task is to monitor correlations among different sensors over time and identify anomalies.
- *Cluster Management* monitors many metrics (such as CPU and memory utilization, disk space, number of processes, etc) of a group of machines. The task is to find main trends and identify anomalies or potential failures.
- *Social Network Analysis* observes an evolving network on social activities (such as citation network). The task is find communities and abnormal individuals.
- *Network Forensics* monitors the Internet communication in the form of (source, destination, port, time, number of packets). The task is to summarize the main communication patterns and identify the attacks and anomalies;
- *Financial Fraud Detection* examines transactional activities of a company over time and tries to identify the abnormal/fraudulent behaviors.

**Data Model:**

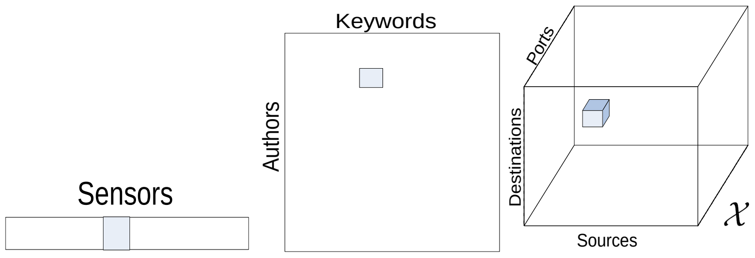
To deal with the diversity of data, we introduce an expressive data model *tensor* from multi-linear analysis [10]. For the Sensor Networks example, we have one measurement (e.g., temperature) from each sensor every timestamp, which forms a high dimensional vector (first order tensor) as shown in Figure 1(a). For the Social Network Analysis, we have authors publishing papers, which forms graphs represented by matrices (second order tensors). For the network forensics 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(c)). An entry ($i$, $j$, $k$) in that tensor (like the small blue cube in Figure 1(c)) has the number of packets from the corresponding source $i$ to the destination $j$ through port $k$, during the given time period.

Focusing on incremental applications, we propose the *tensor stream* (TS) which is an unbounded sequence of tensors. The streaming aspect comes from the fact that new tensors are arriving continuously.

**Incremental Pattern Discovery**:

Incremental Pattern discovery is an online summarization process. In this thesis, we focus on incrementally identifying low-rank structures of the data as the *patterns* and monitor them over time. In another words, we consider the incremental pattern discovery as an incremental dimensionality reduction process.

Let us illustrate the main idea through the network forensics application. In this example, the hourly communication data are represented by high dimensional (3rd order) tensors, which are summarized as low dimensional (3rd order) core tensors in a different space specified by the projections (see Figure 2).



(a) 1st-order (b) 2nd-order (c) 3rd-order

Figure 1: Tensor examples: The blue region indicates a single element in the tensor such as a measurement from a single sensor in (a), the number of papers that an author wrote on a given keyword in (b), the number of packets sent from a source IP to a destination IP through a certain port in (c).

Moreover, the projections capture the overall correlations or hidden variables along three aspects: *source, destination and port*. For example, the *Source projection* characterizes the client correlations; the *Destination projection* summarizes the server correlations; the *Port projection* monitors the port traffic correlations. The projections are dynamically monitored over time.

Furthermore, the core tensor indicates the association across different aspects. More specifically, if there are 3 source hidden variables, 5 destination hidden variables and 6 port hidden variables, the core tensor is a 3-by-5-by-6 3D array, in which the values correspond to the level of association across three different aspects active at different time. More details are covered in Section 3.2.4

For example some web-server hidden variable is always associated with some client behavior through port 80 since those clients always talk to those web servers. In that case, we will see a high value in the corresponding element in the core tensor. Note that this association is dynamic, namely, we may observe different association over time.

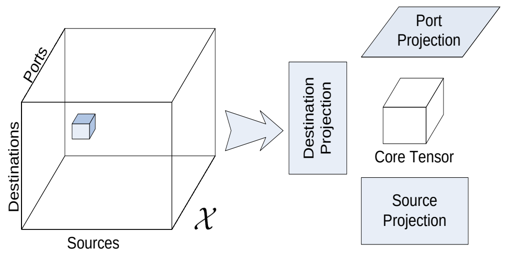


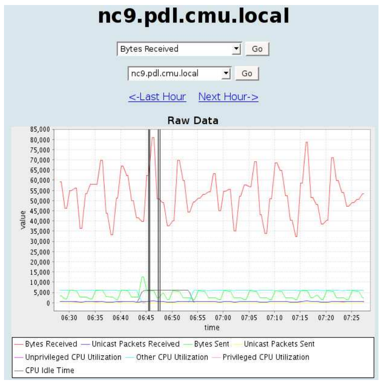
Figure 2: Pattern discovery on a 3rd order tensor

The incremental aspect of the algorithms arrives from the fact that model needs to be constantly updated. More specifically, the problem we study is the follows: Given a stream of tensors $\mathcal{X}_1 \ldots \mathcal{X}_n$, how to compress them incrementally and efficiently? How to find patterns and anomalies? We plan to address two aspects of incremental pattern discovery:

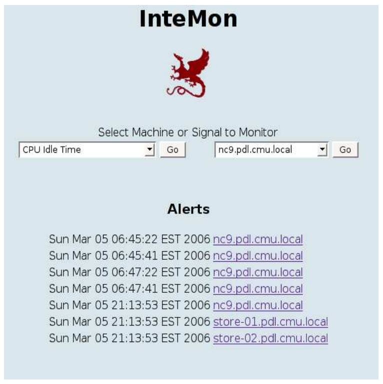
- *Incremental update*: We want to update the old model efficiently, when a new tensor arrives. The key is to avoid redundant computation and storage.

- *Model efficiency* We want an efficient method in term of computational cost and storage consumption. The goal is to achieve linear computation and storage requirement to the update size.

For example, Figure 3 shows the monitoring streams on a storage server which is an example of a first order tensor stream. Each vertical slice correspond a first order tensor (vector). The goal is to find main trend and anomalies as reported on Figure 3(b).



(a) sensor measurement streams



```
Sun Mar 05 06:45:22 EST 2006 nc9.pdl.cmu.local
Sun Mar 05 06:45:41 EST 2006 nc9.pdl.cmu.local
Sun Mar 05 06:47:22 EST 2006 nc9.pdl.cmu.local
Sun Mar 05 06:47:41 EST 2006 nc9.pdl.cmu.local
Sun Mar 05 21:13:53 EST 2006 nc9.pdl.cmu.local
Sun Mar 05 21:13:53 EST 2006 store-01.pdl.cmu.local
Sun Mar 05 21:13:53 EST 2006 store-02.pdl.cmu.local
```

(b) anomalies detected over time

Figure 3: InteMon [21]: automatic cluster monitoring system

**Why it is useful?**:

The result from incremental pattern discovery can be used for many important tasks:

- *Compression*: The core tensor capture most of the information of the original data but in a much lower dimension.
- *Anomaly detection*: From the core tensor, we can approximate the original tensor and compute the reconstruction error. A large reconstruction error is often an indicator for an anomaly.
- *Clustering*: We can often cluster the original data based the projection (see Section 3.2.4 for details). The idea is closely related to latent semantic indexing (LSI) [29].

More importantly, all these tasks can be done incrementally which is essential to many monitoring applications as listed in the beginning of this section.

**Thesis scope**:

Our current work concerns two aspects of the incremental pattern discovery: First, *tensor order* varies from one [30, 33], two [35] to higher order [34]. The complexity [1] increases dramatically as going to higher order tensor. Second, we exploit different *subspace formation* strategies: 1) *orthogonal projection*, which forms orthogonal matrices based on the data (such as SVD, PCA) [30, 33, 34], 2) *example-based projection*, which select judiciously examples from data to form the subspace [35].

Our proposed work includes:

[1]Not only the space and time complexity, but the convergence property increases with the tensor order

| 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 $i$th mode ($1 \leq i \leq M$) |

Table 1: Description of notation.

- *Effective example-based projection*: The example-based method uses the actual data instead of some other abstract notion like orthogonal basis. It has the advantages of intuitive interpretability and sparse encoding. One problem is that the current method [13] sometimes gives a much worse projection (in term of reconstruction error) according to our empirical study than the orthogonal projection such as SVD. We plan to develop a robust method for doing the example-based projection.

- *Incremental example-based projection*: We plan to further extend the exmaple-based method for streaming applications. The goal is to smartly reuse the old model to reduce the computational cost and construct more robust model when new data arrive.

- *Other divergence and distribution*: Currently, we only use the $L_2$ (Euclidean) distance, which assumes the Gaussian distribution. However, the Gaussian distribution may not be a good assumption for many realistic scenarios where nonnegativity and heavy-tailed distribution are required. To address that, we propose to generalize our methods for other divergence such as the Bregman divergence [6].

- *Extensive case study*: To validate the practical value of our methods, we plan to apply the methods on several real applications such as network forensics, cluster monitoring and financial application. We will try to have in-depth study and collaboration with the domain experts in those fields in order to develop practical algorithms.

The layout of this thesis proposal is as follows: Section 2 discusses the related work. Section 3 presents the completed work, followed by the proposed work in Section 4. Finally we conclude in Section 5.

## 2 Survey

In this section, we first discuss some related work in low rank approximation. Second, we introduce multi-linear analysis specially tensor operations. Finally, we present the study on stream and graph mining which are two special cases under the tensor model.

## 2.1 Low rank approximation

### 2.1.1 Singular value decomposition (SVD)

SVD of matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ is a factorization $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ where $\mathbf{U}$ and $\mathbf{V}$ are orthogonal and $\mathbf{\Sigma}$ is diagonal. In addition, the entries of $\mathbf{\Sigma}$ are nonnegative in nonincreasing order. The best $k$ rank approximation can be computed as $\mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k$ where $\mathbf{\Sigma}_k$ is the top left $k$-by-$k$ sub-matrix of $\mathbf{\Sigma}$, and $\mathbf{U}_k$ and $\mathbf{V}_k$ are the first $k$ columns of $\mathbf{U}$ and $\mathbf{V}$ respectively.

SVD has served as a building block for many important applications, such as PCA [23] and LSI [29, 11], and has been used as a compression technique [25]. It has also been applied as correlation detection routine for streaming settings [18, 30].

**Pros**: SVD is often used as a dimensionality reduction tool for matrices with low rank structures (i.e., $k \ll \min(m, n)$). The computed singular vectors are also useful for tasks such as clustering and outlier detection.

**Cons**: When matrix $\mathbf{A}$ is sparse, it can be stored in sparse matrix representation so that the space requirement is proportional to the number of nonzero entries. Unfortunately, after SVD, $\mathbf{U}$ and $\mathbf{V}$ become dense due to the orthogonalization process. Although the new matrix dimension $k$ is still small for low rank approximation, the total space for storing $\mathbf{U}$ and $\mathbf{V}$, may become much larger than the space required for storing $\mathbf{A}$. Even worse, if the dimensions of $\mathbf{A}$ are large($m$ and $n$ are large), SVD becomes computationally too expensive to perform.

### 2.1.2 Principal component analysis (PCA)

As shown in Figure 4, PCA finds the best linear projections of a set of high dimensional points that minimizes 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[2].

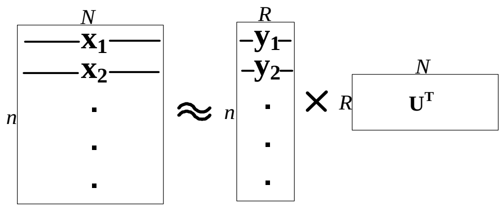


Figure 4: PCA projects the $N$-D vector $\mathbf{x}_i$s into $R$-D vector $\mathbf{y}_i$s and $\mathbf{U}$ is the projection matrix.

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 $\mathbf{U} = \mathbf{V}_{svd}$

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

### 2.1.3 CUR decomposition

Drineas et al. [13] proposed a powerful method, called CUR decomposition, to perform matrix factorization.

Such method approximates the input matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ as a product of three small matrices constructed from sampled columns and rows, while preserving the sparsity of the original $\mathbf{A}$ after decomposition. More formally, it approximates matrix $\mathbf{A}$ as $\tilde{\mathbf{A}} = \mathbf{CUR}$, where $\mathbf{C} \in \mathbb{R}^{m \times c}$ ($\mathbf{R} \in \mathbb{R}^{r \times n}$) contains $c(r)$ scaled columns(rows) sampled from $\mathbf{A}$, and $\mathbf{U} \in \mathbb{R}^{c \times r}$ is a small dense matrix which can be computed from $\mathbf{C}$ and $\mathbf{R}$.

The key idea of CUR is to sample matrix rows and columns strategically, with replacement biased towards those ones with higher norms (see Figure 5). In other words, the columns and rows with higher entry values will have higher chance to be selected multiple times. Once we obtain a set of sampled rows $\mathbf{R}$ and columns $\mathbf{C}$, we can compute $\tilde{\mathbf{A}}$ as the projection of $\mathbf{A}$ onto the span of $\mathbf{C}$.

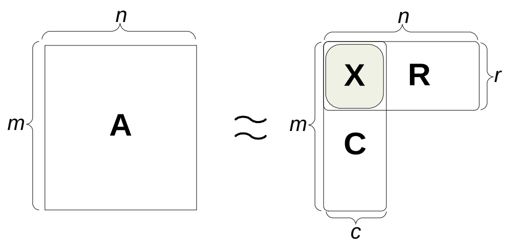


Figure 5: CUR approximates matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ by the product of three matrices $\mathbf{C}$, $\mathbf{U}$, $\mathbf{R}$ where $\mathbf{C}$ and $\mathbf{R}$ are sampled columns and rows, $\mathbf{U}$ is the pseudo-inverse of $\mathbf{X}$ (the intersection of $\mathbf{C}$ and $\mathbf{R}$)

**Pros**: CUR provides an efficient approximation of SVD. It requires only two passes over the input matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ with a small buffer size on the order of $O(m+n)$. Unlike SVD, the resulting matrices from CUR are still sparse (except for the matrix $\mathbf{U} \in \mathbb{R}^{c \times r}$ which is not necessarily stored since it can be computed from $\mathbf{C}$ and $\mathbf{R}$). Because CUR does not require orthogonalization on the entire matrix $\mathbf{A}$, it is also faster in computation than SVD.
**Cons**: The biased sampling with replacement usually results in many duplicate columns and rows in $\mathbf{C}$ and $\mathbf{R}$, especially for input graphs that follow power-law distributions. On the one hand, these duplicates are necessary for the algorithm to achieve a low sum of squared error (SSE). Furthermore, simply ignoring the duplicate samples does not solve the problem and usually brings huge error. On the other hand, they do incur repeated computation and additional space requirement.

## 2.2 Multilinear Analysis

### 2.2.1 Tensor Basics

A tensor of order $M$ closely resembles a Data Cube with $M$ dimensions. Formally, we write an $M$th order tensor $\mathcal{X} \in \mathbb{R}^{N_1 \times \cdots \times N_M}$ as $\mathcal{X}_{[N_1,\ldots,N_M]}$, where $N_i$ ($1 \leq i \leq M$) is the *dimensionality* of the $i$th 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 $M$th 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 $\mathbb{R}^{(\prod_{i \neq d} N_i) \times N_d}$.*

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 $\mathcal{X} = \text{fold}(\text{unfold}(\mathcal{X}, d))$. Figure 6 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 6 the slice of the 3rd mode along the 2nd dimension.

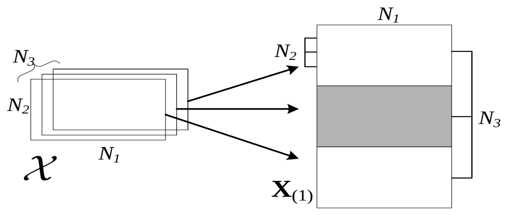


Figure 6: 3rd order tensor $\mathcal{X} \in \mathbb{R}^{N_1 \times N_2 \times N_3}$ is matricized along mode-1 to a matrix $\mathbf{X}_{(1)} \in \mathbb{R}^{(N_2 \times N_3) \times N_1}$. The shaded area is the slice of the 3rd mode along the 2nd dimension.

**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:*

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

*for all index values.*

The *Mode Product* for 2nd order tensor degenerates to matrix product. Figure 7 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.

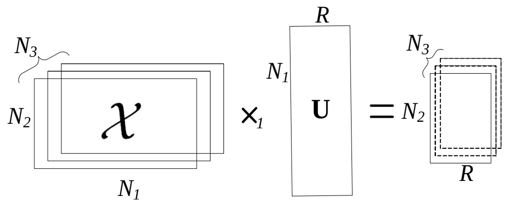


Figure 7: 3rd order tensor $\mathcal{X}_{[N_1,N_2,N_3]} \times_1 \mathbf{U}$ results in a new tensor in $\mathbb{R}^{R \times N_2 \times N_3}$

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 $\mathcal{X} \prod_{j \neq i} \times_j \mathbf{U}_j$.

**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 $\operatorname{rank}\left(\tilde{\mathbf{X}}_{(d)}\right) = R_d$ for $1 \leq d \leq M$, 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 $\mathcal{X}$.*[3]

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 $\mathbf{U}_l|_{l=1}^{M} \in \mathbb{R}^{N_l \times R_l}$.

### 2.2.2 Tensor Decomposition

Tensor algebra and multilinear analysis have been applied successfully in many domains [10, 24, 38]. Powerful tools have been proposed, including Tucker decomposition [36], parallel factor analysis [19] or canonical decomposition [7]. Tensors have been recently used in machine vision research, for example by Shashua and Levin [31] for linear image coding, by Vasilescu and Terzopoulos [37] for face recognition. Ye [42] presented the generalized low rank approximations which extends PCA from the vectors (1st-order tensors) into matrices (2nd order tensors). Ding and Ye [12] proposed an approximation of [42]. Similar approach is also proposed in [20]. Xu et al. [40] formally presented the tensor representation for PCA and applied it for face recognition. Drineas and Mahoney [15] 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.

## 2.3 Stream Mining

Data streams has been extensively studied in recent years. The goal is to process the incoming data efficiently without recomputing from scratch and without buffering much historical data. Two recent surveys [3, 28] have discussed many data streams algorithms, among which we highlight two related techniques: sampling and sketches.

Sampling is a simple and efficient method to deal with large massive datasets. Many sampling algorithms have been proposed in the streaming setting such as reservoir sampling [39], concise samples, and counting samples [17]. These advanced sampling techniques are related to our example-based projection technique.

"Sketch" is another powerful technique to estimate many important statistics, such as $L_p$-norm [22, 8],of an unbounded stream using a compact structure. The underlying theory of "sketch" is to perform dimensionality reduction using random projections as opposed to the best-$k$ rank approximations. Random

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

projection has the advantage that it is fast to compute and preserves the distance between nodes. However, the projection leads to dense data representation.

### 2.4 Graph Mining

Graph mining has been a very active area in data mining community. Because of its importance and expressiveness, various problems are studied under graph mining.

From the explorative aspect, Faloutsos et al. [16] have shown the powerlaw distribution on the Internet graph. Kumar et al. [26] studied the model for web graphs. Leskovec et al. [27] discoverd the shrinking diameter phenomena on time-evolving graphs.

From the algorithmic aspect, Yan et al. [41] proposed an algorithm to perform substructure similarity search on graph databases, which is based on the algorithm for classic frequent itemset mining. Cormode and Muthukrishnan [9] proposed streaming algorithms to (1) estimate frequency moments of degrees, (2) find heavy hitter degrees, and (3) compute range sums of degree values on streams of edges of communication graphs, i.e., (source, destination) pairs. In our work, we view graph mining as a matrix decomposition problem (2nd order tensor) and try to approximate the entire graph, which is different from most of the existing graph mining work.

## 3 Current Work

Our current work consists of the following parts:

- *SPIRT and distributed SPIRIT* [30, 33]: SPIRIT is a streaming algorithm for finding patterns in multiple co-evolving streams (first order tensor streams). We further extend SPIRIT into a distributed algorithm where distributed streams are assigned into different groups, then local patterns are computed from each group, and finally global patterns are constructed from all local patterns. In addition to the experiments on several real datasets, we implemented two prototype systems on wireless sensor Motes [32] and machine monitoring [21], respectively.

- *Dynamic and Streaming Tensor Analysis* [34]: We proposed tensor streams as a general and expressive data model for many different applications. Two incremental algorithms are proposed, namely, dynamic tensor analysis (DTA) and streaming tensor analysis (STA).

- *Compact Matrix Decomposition (CMD)* [35]: Observing the sparsity in many real data, we studied an alternative decomposition method to SVD, which preserves the sparsity property and provides more intuitive results.

### 3.1 SPIRIT: incremental pattern discovery for 1st order tensor streams

Given a collection of $N$ streams, SPIRIT does the following:

- Adapts the number of $k$ main trends (hidden variables) to summarize the $N$ streams.

- Adapts the projection matrix $\mathbf{U}$ which determines the participation weights of each streams on a hidden variable.

More formally, the collection of streams is $\mathbf{X} \in \mathbb{R}^{n \times N}$ where 1) every row is a $N$-dimensional vector containing values at a certain timestamp and 2) $n$ is increasing and unbounded over time; SPIRIT finds $\mathbf{X} = \mathbf{Y}\mathbf{U}^T$ incrementally where the hidden variable $\mathbf{Y} \in \mathbb{R}^{n \times R}$ and the projection matrix $\mathbf{U} \in \mathbb{R}^{N \times R}$. In a sensor network example, at every time tick there are $N$ measurements each from a temperature sensor in Wean hall; These $N$ measurements (one row in matrix $\mathbf{X}$) map to $R$ hidden variables (one row in matrix $\mathbf{Y}$) through the projection matrix $\mathbf{U}$. An additional complication is that $\mathbf{U}$ is changing over time based on the recent values from $\mathbf{X}$.

### 3.1.1 Monitoring variance matrix

Let us consider the offline version first where $\mathbf{X}$ is bounded and static. Given a matrix $\mathbf{X} \in \mathbb{R}^{n \times N}$, the variance matrix is defined as $\mathbf{C} = \mathbf{X}^T\mathbf{X}$. And the diagonalization of $\mathbf{C}$ gives us the projection matrix $\mathbf{U}$, namely, $\mathbf{C} = \mathbf{U}\mathbf{\Sigma}\mathbf{U}^T$. The hidden variables $\mathbf{Y} = \mathbf{X}\mathbf{U}$.

For the online version when new values arrive as a matrix $\mathbf{X}_{new} \in \mathbb{R}^{n' \times N}$ [4], the variance matrix can be updated as $\mathbf{C}_{new} = \mathbf{C} + \mathbf{X}_{new}^T\mathbf{X}_{new}$. And the new projection matrix can be obtained by diagonalizing $\mathbf{C}_{new}$. Again, the hidden variable $\mathbf{Y}_{new} = \mathbf{X}_{new}\mathbf{C}_{new}$.
**Complexity**: Assuming $\mathbf{X}_{new}$ is a vector, the computation complexity is $O(RN^2)$ and the space complexity $O(N^2)$ where $R$ is the number of hidden variables and $N$ the number of streams.

### 3.1.2 SPIRIT

The previous algorithm is simple and exact but still too expensive for some application when the number of streams $N$ is large. We proposed SPIRIT, an approximation achieving linear space and computation cost in $N$.
**Tracking a projection matrix**: The diagonalization process in the previous algorithm can be expensive. Especially, when the change of the variance matrix is small, it is not worth diagonalizing that matrix. The idea is to continuously track the changes of projection matrices using the recursive least-square technique for sequentially estimating the principal components.

The main idea behind the tracking algorithm is to read in a new vector $\mathbf{x}$ and perform three steps:

1. Compute the projection $\mathbf{y}$ by projecting $\mathbf{x}$ onto $\mathbf{U}$;
2. Estimate the reconstruction error ($\mathbf{e}$) and the energy (the sum of squares of all the past values), 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 values. 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* (the sum of squares of all the previous values).
**Complexity**: The computation and space complexity are $O(NR)$.
**Detecting the number of hidden variables $R$**:

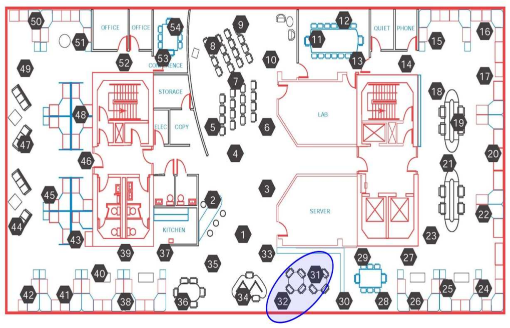
[4] When $n' = 1$, $\mathbf{X}_{new}$ becomes a vector which is equivalent to add one more row to original matrix $\mathbf{X}$.

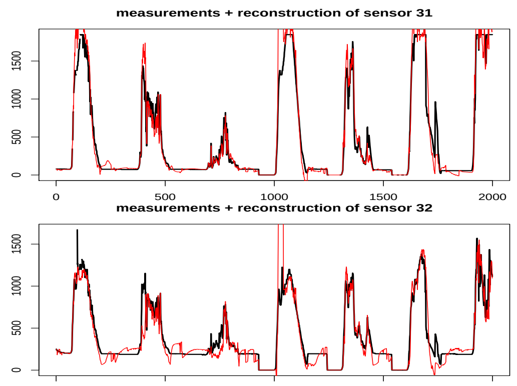
We use the Energy thresholding[5] to deterimine the number $R$. The idea is to increase or decrease the number of hidden variables when the ratio between the energy kept by the hidden variables and the one kept by the input values is below or above a certain threshold (e.g. .95 and .98 in our experiments).

**Other extensions**:

In addition to spotting the correlations across different streams, SPIRIT can also help to provide the following functionalities:

- *Forgetting factor*: We can adapt to more recent behavior by using an exponential forgetting factor $0 < \lambda < 1$ to the old energy. This allows us to follow trend drifts over time.
- *Forecasting*: We can apply any forecasting algorithm on the hidden variables $\mathbf{Y}$ instead of original values $\mathbf{X}$. This is a much better method for forecasting, because 1) *efficiency*: the number of hidden variables is much smaller than the original streams; 2) *effectiveness*: a simple forecasting method like autoregression may work well since the hidden variables are uncorrelated by construction (same reason as PCA).
- *Missing values*: When we have a forecasting model, we can use the forecast to estimate missing values. We then use these estimated missing values to update the projection matrix $\mathbf{U}$, as well as the forecasting model.
- *Distributed streams*: We present a hierarchical approach in [33] to deal with distributed streams. This method avoids many drawbacks of the centralized approach such as single point of failure, communication bottleneck, and scalability issue. It requires limited shared information across different groups.


(a) Lab map



(b) Original measurements vs. reconstruction

Figure 8: `Mote` dataset: it shows the measurements (bold) and reconstruction (thin) on node 31 and 32 (highlighted in (a)).

[5]It is a common method to determine how many principal components are needed [23].

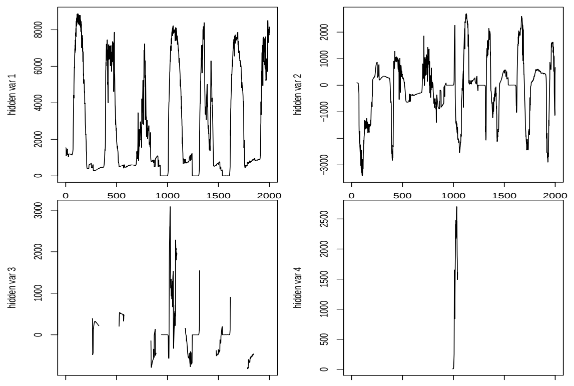


Figure 9: Hidden variables: The third and fourth hidden variables are intermittent and indicate “anomalous behavior”. Note that the axes limits are different in each plot.

### 3.1.3 Experiment

We did extensive experiments on many real datasets for both centralized and distributed scenarios [30, 33]. And we implemented two systems using this algorithm on actual Motes sensors [32] and cluster monitoring [21]. Here we present one experiment to illustrate the point.
**Description** The `Motes` light dataset consists of 48 light intensity measurements collected using Berkeley Mote sensors, at several different locations in a lab (see Figure 8), over a period of a week (2000 timestamps).
**Data characteristics** The main characteristics are:

- A clear global periodic pattern (daily cycle).
- Occasional big spikes from some sensors (outliers).

**Results of SPIRIT** SPIRIT detects four hidden variables (see Figure 9). Two of these are intermittent and correspond to outliers, or changes in the correlated trends. We show the reconstructions for some of the observed variables in Figure 8(b).
**Interpretation** In summary, the first two hidden variables (see Figure 9) correspond to the global trends and the last two, which are intermittently present, correspond to outliers. In particular:

- The first hidden variable captures the global periodic pattern.
- The interpretation of the second one is due to the phase-shift across different streams. The first two hidden variables together are sufficient to express arbitrary phase shifts.
- The third and fourth hidden variables indicate some of the potential outliers in the data. For example, there is a big spike in the 4th hidden variable at time $t = 1033$, as shown in Figure 9. Examining the corresponding participation weights in $\mathbf{U}$ at that timestamp, we can find the corresponding sensors “responsible” for this anomaly, i.e., those sensors whose participation weights have very high magnitude. Among these, the most prominent are sensors 31 and 32. Looking at the actual measurements from these sensors, we see that before time $t = 1033$ they are almost 0. Then, very large increases occur around $t = 1033$, which bring an additional hidden variable into the system.

## 3.2 DTA and STA: incremental pattern discovery for high order tensor streams

### 3.2.1 Problem Definition

Here we first formally define the *tensor streams*. Then we overview the operations on them.

**Definition 4 (Tensor stream)** *A sequence of $M$th order tensor $\mathcal{X}_1 \dots \mathcal{X}_n$, where each $\mathcal{X}_i \in \mathbb{R}^{N_1 \times \dots \times N_M}$ ($1 \le i \le n$), is called a tensor stream if $n$ is a positive integer that increases with time.*

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

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

**Definition 5 (Tensor analysis)** *Given a sequence of tensors $\mathcal{X}_1 \dots \mathcal{X}_n$, where each $\mathcal{X}_i \in \mathbb{R}^{N_1 \times \dots \times N_M}$ ($1 \le i \le n$), 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 $\mathcal{Y}_t \prod_{i=1}^{M} \times_i \mathbf{U}_i^T$ where $\mathcal{Y}_t$ is the core tensor defined as $\mathcal{Y}_t = \mathcal{X}_t \prod_{i=1}^{M} \times_i \mathbf{U}_i$ (see Figure 12 for the intuition behind).

More specifically, we propose two variants under *tensor analysis*: dynamic tensor analysis (DTA) and streaming tensor analysis (STA) for a tensor stream.

### 3.2.2 Dynamic Tensor Analysis

Here we present the dynamic tensor analysis (DTA), an incremental algorithm for tensor dimensionality reduction.

**Intuition:** The idea of the incremental algorithm is to exploit two facts:

1. In general *tensor analysis* can be computed relatively quickly once the variance matrices [6]are available;

2. Variance matrices can be incrementally updated without storing any historical tensor as shown in Section 3.1.1.

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

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

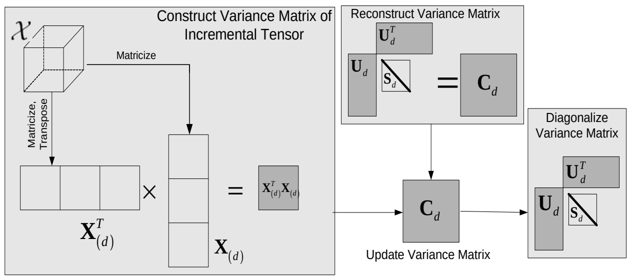

Construct Variance Matrix of Incremental Tensor
Matricize
Matricize, Transpose
Reconstruct Variance Matrix
Diagonalize Variance Matrix
Update Variance Matrix

Figure 10: New tensor $\mathcal{X}$ is matricized along the $d$th 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 $\mathbf{X}_{(d)} \in \mathbb{R}^{(\prod_{i\neq d} N_i)\times N_d}$ 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 process is visualized in Figure 10.

Similar to the SPIRIT, we incorporate the *forgetting factor* to emphasize the recent data and use the *energy criterion* to estimate the number of hidden variables along each mode.

**Complexity:** The space consumption for the incremental algorithm is $\prod_{i=1}^M N_i$ + $\sum_{i=1}^M N_i \times R_i$ + $\sum_{i=1}^M R_i$ where $N_i$ is the dimensionality of the $i$th mode and $R_i$ the number of hidden variables along the $i$th mode (the rank along the $i$th mode). The dominant factor is from the first term O($\prod_{i=1}^M N_i$). However, standard offline tensor analysis requires O($n\prod_{i=1}^M N_i$) for storing all tensors up to time $n$, which is unbounded.

The computation cost is $O(\sum_{i=1}^M R_i N_i^2 + \prod_{i=1}^M N_i$. Note that for medium or low mode tensors (i.e., order $M \leq 5$), the diagonalization ($O(\sum_{i=1}^M R_i N_i^2)$) is the main cost. Section 3.2.3 introduces a faster approximation of DTA that avoids diagonalization.

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

### 3.2.3 Streaming Tensor Analysis

Now we present the *streaming tensor analysis* (STA), a fast algorithm to approximate DTA without diagonalization.

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 $\mathbf{X}_{(d)}$, then adjust the projection matrix $\mathbf{U}_d$ by applying SPIRIT algorithm (see Section 3.1) over the rows of $\mathbf{X}_{(d)}$. 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 the same as DTA, 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 DTA (when $R_i \ll N_i$). The STA

[6]Recall the variance matrix along the $d$th mode of $\mathbf{X}_{(d)} \in \mathbb{R}^{(\prod_{i\neq d} N_i)\times N_d}$ is defined as $\mathbf{C} = \mathbf{X}_{(d)}^T\mathbf{X}_{(d)} \in \mathbb{R}^{N_d\times N_d}$.

can be further improved with random sampling technique, i.e., use only subset of rows of $\mathbf{X}_{(d)}$ for update.

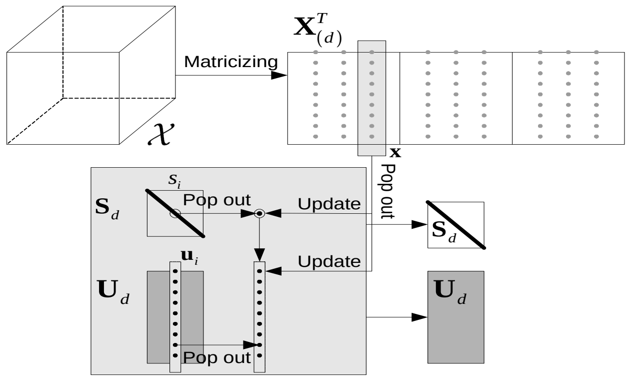


Figure 11: New tensor $\mathcal{X}$ is matricized along the $d$th mode. For every row of $\mathbf{X}_d$, we update the projection matrix $\mathbf{U}_d$. And $\mathbf{S}_d$ helps determine the update size.

### 3.2.4 Key applications

We investigate the following two applications of DTA and STA:
***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 streams: 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.

DTA/STA enables us to quantify the multi-level anomaly score through a simple reconstruction error.
***Multi-way latent semantic indexing***: The goal of the multi-way LSI is to find highly 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).

### 3.2.5 Experiment

**Network data**: 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. We partition raw data stream into disjoint hourly based windows and construct a tensor for each window.

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

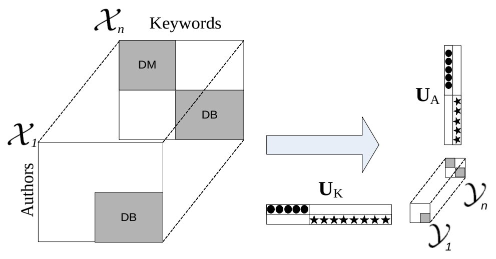


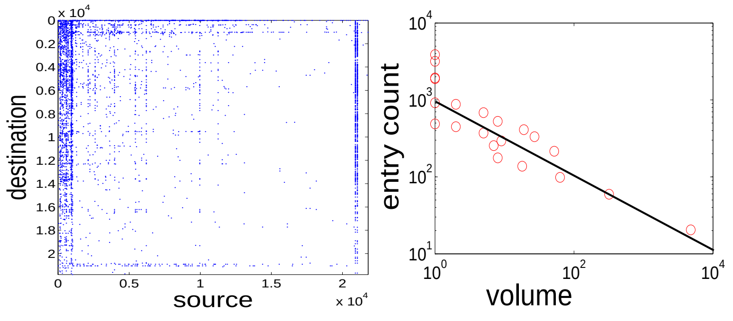
Figure 12: $\mathbf{U}_A$ and $\mathbf{U}_K$ capture the DB (stars) and DM (circles) concepts in authors and keywords, respectively; initially, only DB is activated in $\mathcal{Y}_1$; later on both DB and DM are in $\mathcal{Y}_n$.

| **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

high traffic. Figure 14(a) shows an example source-destination matrix constructed using traffic data generated from 10AM to 11AM on 01/06/2005. We observe that the matrix is indeed sparse, with most of the traffic to or from a small set of server like hosts.

Moreover, the distribution of the entry values is very skewed (a power law distribution) as shown in Figure 14(b). Most of hosts have zero traffic, with only a few of exceptions which were involved with high volumes of traffic (over $10^4$ flows during that hour). Given such skewed traffic distribution, we re-scale all the non-zero entries by taking the natural logarithm (actually, $\log(x+1)$, to account for $x = 0$), so that the matrix decomposition results will not be dominated by a small number of very large entry values.



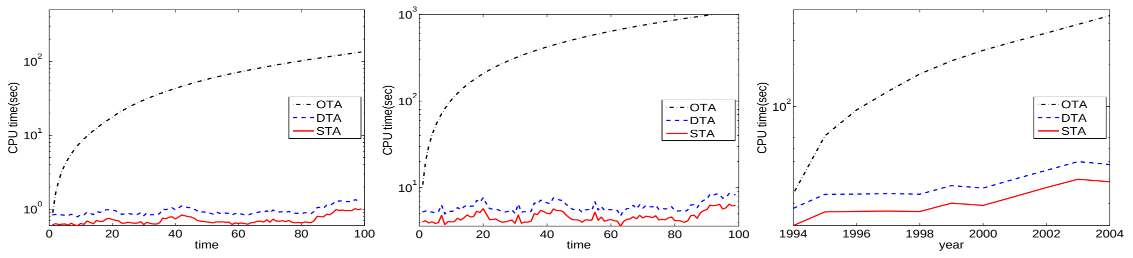
(a) Source-destination matrix (b) Entry distribution

Figure 14: Network Flow: the example source-destination matrix is very sparse but the entry values are skewed.

**Bibliographic data**: 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 simple stemming and stop-word removal.

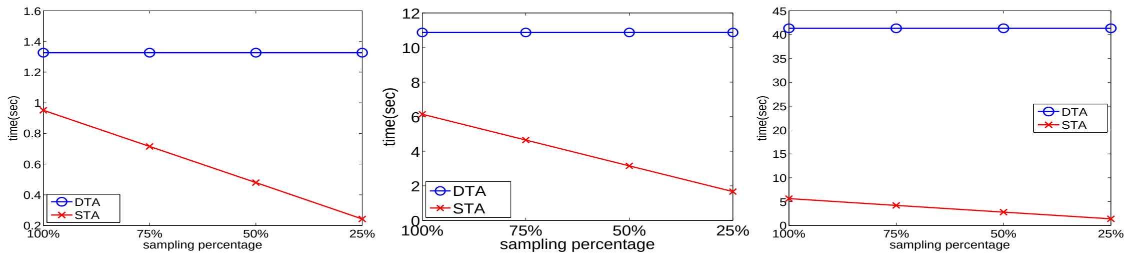
**Computational cost:** We first compare three different methods, namely, offline tensor analysis (OTA), dynamic tensor analysis (DTA), streaming tensor analysis (STA), in terms of computation time for different datasets. Figure 15 shows the CPU time in logarithm as a function of relapse time. Since the new tensors keep coming, the cost of OTA increases linearly[7]; while DTA and STA remains more or less constant. Note that DBLP in Figure 15(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.



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

Figure 15: Both DTA and STA use much less time than OTA over time across different datasets

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 16 shows the CPU time vs. sampling rate, where STA runs much faster compared to DTA.



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

Figure 16: STA uses much less CPU time than DTA across different datasets

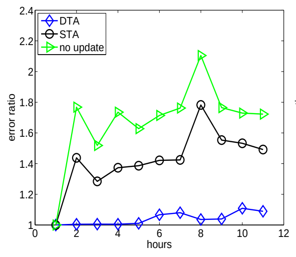
**Accuracy comparison:** Now we evaluate the approximation accuracy of DTA and STA compared to OTA.

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

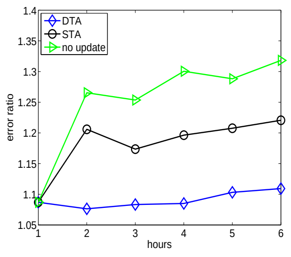
**Performance metric:** Intuitively, the goal is to be able to compare how accurate each tensor decomposition is to the original tensors. Therefore, reconstruction error provides a natural way to quantify the accuracy. Recall the reconstruction error is defined in Definition 5. Error can always be reduced 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 DTA and STA are close to the expensive OTA method (see Figure 17(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 17 since after that point OTA runs out of the memory.

In more details, Figure 17(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.

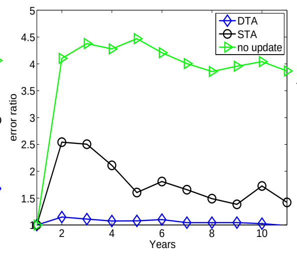
DTA performs very close to OTA, which suggests a cheap incremental methods over the expensive OTA. The even cheaper method, STA, usually gives good approximation to DTA (see Figure 17(a) and (b) for IP2D and IP3D). But note that STA performs considerably worse in DBLP in Figure 17(c) because the adaptive 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 tensors with significant time-dependency (i.e., the changes over consecutive timestamps should not be too big).



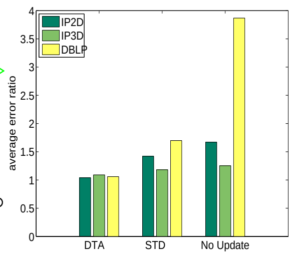
(a) `IP2D`



(b) `IP3D`



(c) `DBLP`



(d) Average

Figure 17: Reconstruction error over time

**Case study I: Anomaly detection on Network traffic**

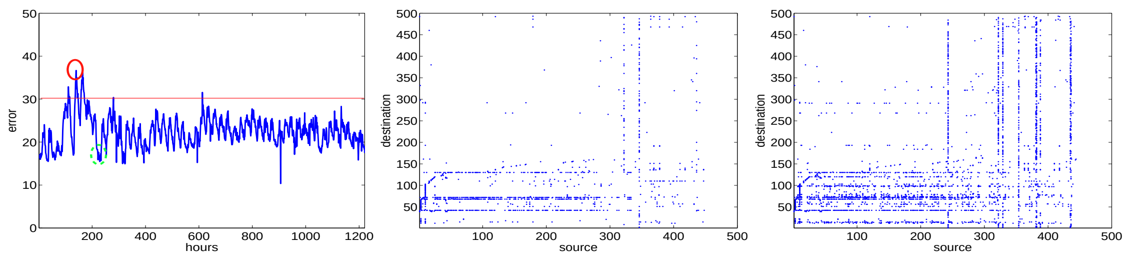
For network traffic, normal host communication patterns in a network should roughly be similar to each other over time. A sudden change of approximation accuracy suggests structural changes of communication patterns since the same approximation procedure can no longer keep track of the overall patterns. Figure 18(a) shows the relative reconstruction error[8] 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 18(a). We manually investigate into the trace further, and indeed find the onset of worm-like hierarchical scanning activities. Figure 18 (b) and (c) shows the normal (green dash circle in (a)) and abnormal (red solid circle in (a)) timestamps. The dot in Figure 18 means there are packet flows between the corresponding source and destination. The prominent difference between these two is mainly due to the unusual

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

| 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 2: Example clusters: first two lines databases groups, last line data mining group. Note that keywords are after stemming

scanning activities (more columns with many dots). We can successfully identify this real anomaly in the data using reconstruction error from DTA.



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

Figure 18: Network flow over time: we can detect anomalies by monitoring the approximation accuracy

**Case study II: Multi-way LSI** Similarly, we perform 2-way LSI on DBLP datasets over time. Two example clusters are listed in Table 2. The algorithm correctly separates the two groups (data mining, databases) of people and concepts (2nd and 3rd clusters). Furthermore, 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).

## 3.3 Compact Matrix Decomposition for large sparse graphs

Up to now, we only study orthogonal projection like what SVD or PCA do. Although these methods are very successful in general, for large sparse data they tend to require huge amounts of space, exactly because their resulting matrices are not sparse any more. Another drawback of an orthogonal projection is the lack of an intuitive interpretation.

Recently, Drineas et al. [13] proposed the CUR decomposition method, which decomposes a large sparse matrix into three smaller matrices by judiciously sampling columns and rows from the original matrix. Although the constructed low-rank approximations may not be optimal, CUR does partially address the loss-of-sparsity issue.

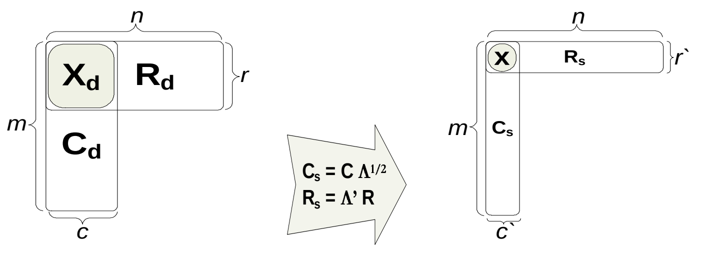
We propose a new method, called *Compact Matrix Decomposition (CMD)*, for generating low-rank matrix approximations. CMD provides provably equivalent decomposition as CUR, but it requires much *less* space and computation time, and hence is *more* efficient. More specifically, CMD approximates the input matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ as a product of three small matrices constructed from sampled columns and rows, while preserving the sparsity of the original $\mathbf{A}$ after decomposition. More formally, it approximates the matrix $\mathbf{A}$ as $\tilde{\mathbf{A}} = \mathbf{C_s U R_s}$, where $\mathbf{C_s} \in \mathbb{R}^{m \times c'}$ ($\mathbf{R_s} \in \mathbb{R}^{r' \times n}$) contains $c(r)$ scaled columns(rows) sampled from $\mathbf{A}$, and $\mathbf{U} \in \mathbb{R}^{c' \times r'}$ is a small dense matrix which can be computed from $\mathbf{C_s}$ and $\mathbf{R_s}$.

### 3.3.1 Subspace Construction

Since the subspace is spanned by the columns of the matrix, we choose to use sampled columns to represent the subspace.

**Biased sampling:** The key idea for picking the columns is to sample columns with replacement biased towards those ones with higher norms. In other words, the columns with higher entry values will have higher chance to be selected multiple times. Such sampling procedure is proved to yield an optimal approximation [13]. Note that, the biased sampling will bring a lot of duplicated samples.

**Duplicate column removal:** CMD carefully removes duplicate columns and rows after sampling, and thus it reduces both the storage space as well as the computational effort. Intuitively, the directions of those duplicate columns are more important than the other columns. Thus a key step of subspace construction is to scale up the columns that are sampled multiple times while removing the duplicates. Pictorially, we take matrix $\mathbf{C}_d$ and turn it into the much narrower matrix $\mathbf{C}_s$ as shown in Figure 19(b), with proper scaling. In [35] we proved that the rescaled distinct columns have the same best $k$ approximation as the columns with duplicates.



(a)with duplicates (b) without duplicates

Figure 19: Illustration of CUR and CMD

### 3.3.2 Low rank approximation

The goal is to form an approximation of the original matrix $\mathbf{X}$ using the sampled column $\mathbf{C}_s$. For clarity, we use $\mathbf{C}$ for $\mathbf{C}_s$. More specifically, we want to project $\mathbf{X}$ onto the space spanned by $\mathbf{C}$, which can be done as follows:

Note that the set of selected columns $\mathbf{C} \in \mathbb{R}^{m \times c}$ do not form an orthonormal basis. One possibility is to use the fact that given an arbitrary basis $\mathbf{B}$ (not necessarily orthonormal), the projection to the span of $\mathbf{B}$ is $\mathbf{B}(\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T$. Unfortunately, although $\mathbf{C}$ specifies the subspace, in general $\mathbf{C}$ may not form a basis because the columns may not be linearly independent for $(\mathbf{C}^T\mathbf{C})^{-1}$ to exist.

We first construct the orthonormal basis of $\mathbf{C}$ using SVD (say $\mathbf{C} = \mathbf{U}_C \mathbf{\Sigma}_C \mathbf{V}_C^T$), and then projecting the original matrix into this identified orthonormal basis $\mathbf{U}_C \in \mathbb{R}^{m \times c}$. Since both $\mathbf{U}_C$ and $\mathbf{U}_C^T$ are usually large and dense, we do not compute the projection of matrix $\mathbf{A}$ directly as $\mathbf{U}_C \mathbf{U}_C^T \mathbf{A}$. Instead, we computes a low rank approximation of $\mathbf{A}$ based on the observation that $\mathbf{U}_c = \mathbf{C}\mathbf{V}_C \mathbf{\Sigma}_C^{-1}$, where $\mathbf{C} \in \mathbb{R}^{m \times c}$ is large but sparse, $\mathbf{V}_C \in \mathbb{R}^{c\times k}$ is dense but small, and $\mathbf{\Sigma} \in \mathbb{R}^{k\times k}$ is a small diagonal matrix [9]. Therefore, we have the following:

$$\begin{aligned}\tilde{\mathbf{A}} &= \mathbf{U}_c\mathbf{U}_c^T\mathbf{A} = \mathbf{C}\mathbf{V}_C\mathbf{\Sigma}_C^{-1}(\mathbf{C}\mathbf{V}_C\mathbf{\Sigma}_C^{-1})^T\mathbf{A}\\ &= \mathbf{C}(\mathbf{V}_C\mathbf{\Sigma}_C^{-2}\mathbf{V}_C^T\mathbf{C}^T)\mathbf{A} = \mathbf{CTA}\end{aligned}$$

where $\mathbf{T} = (\mathbf{V}_C\mathbf{\Sigma}_C^{-2}\mathbf{V}_C^T\mathbf{C}^T) \in \mathbb{R}^{c\times m}$.

Although $\mathbf{C} \in \mathbb{R}^{m\times c}$ is sparse, $\mathbf{T}$ is still dense and big. we further optimize the low-rank approximation by reducing the multiplication overhead of two large matrices $\mathbf{T}$ and $\mathbf{A}$. Specifically, given two matrices $\mathbf{A}$ and $\mathbf{B}$ (assume $\mathbf{AB}$ is defined), we can sample both columns of $\mathbf{A}$ and rows of $\mathbf{B}$ using the biased sampling algorithm (i.e., biased towards the ones with bigger norms). The selected rows and columns are then scaled accordingly for multiplication. This sampling algorithm brings the same problem as column sampling, i.e., there exist duplicate rows. Finally, CMD removes duplicate rows using a different scaling factor. In our context, CMD samples and scales $r'$ unique rows from $\mathbf{A}$ and extracts the corresponding $r'$ columns from $\mathbf{C}^T$ (last term of $\mathbf{T}$).

### 3.3.3 CMD in practice

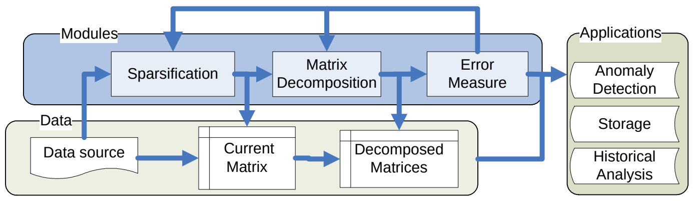


Figure 20: A flowchart for mining large graphs with low rank approximations

Figure 20 shows the flowchart of the whole mining process. The process takes as input data from the application, and generates as output mining results represented as low-rank data summaries and approximation errors. The results can be fed into different mining applications such as anomaly detection and historical analysis.

The *data source* is assumed to generate a large volume of real time event records for constructing large graphs (e.g., network traffic monitoring and analysis). Because it is often hard to buffer and process all data that are streamed in, we propose one more step, namely, *sparsification*, to reduce the incoming data volume by sampling and scaling data to approximate the original full data.

Given the input data summarized as a *current matrix* $\mathbf{A}$, the next step is *matrix decomposition*, which is the core component of the entire flow to compute a lower-rank matrix approximation. Finally, the *error measure* quantifies the quality of the mining result as an additional output.

### 3.3.4 Experiment

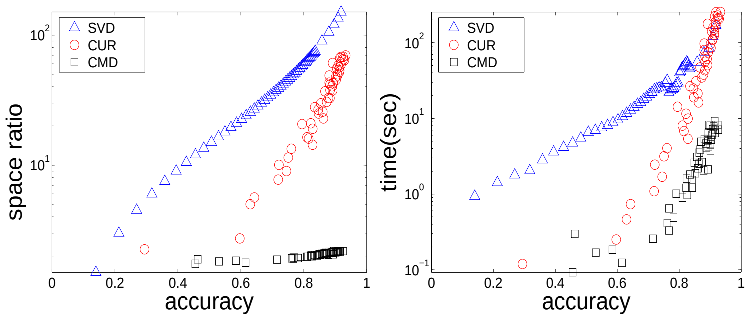
We use the complete network and DBLP data for evaluation without any sampling.

[9] In our experiment, both $\mathbf{V}_C$ and $\mathbf{\Sigma}_C$ have significantly smaller number of entries than $\mathbf{A}$.

| data | dimension | $\lvert E \rvert$ | nonzero entries |
|---|---|---|---|
| Network flow | 22K-by-22K | 12K | $0.0025\%$ |
| DBLP data | 428K-by-3.6K | 64K | $0.004\%$ |

Figure 21: Two datasets

**Performance on Network data**: We first evaluate the space consumption for three different methods to achieve a given approximation accuracy. Figure 22(a) shows the space ratio (to the original matrix) as the function of the approximation accuracy for network flow data. Note the Y-axis is in log scale. Among the three methods, CMD uses the least amount of space consistently. SVD uses the most amount of space (over 100X larger than the original matrix). CUR uses a similar amount of space as CMD to achieve a low accuracy. But when we increase the target accuracy to achieve, the space consumption of CUR increases dramatically (over 50X larger than the original matrix). The reason is that CUR has to keep many duplicate columns and rows in order to reach a high accuracy, while CMD keeps only unique columns and rows.



(a) space vs accuracy (b) time vs. accurarcy

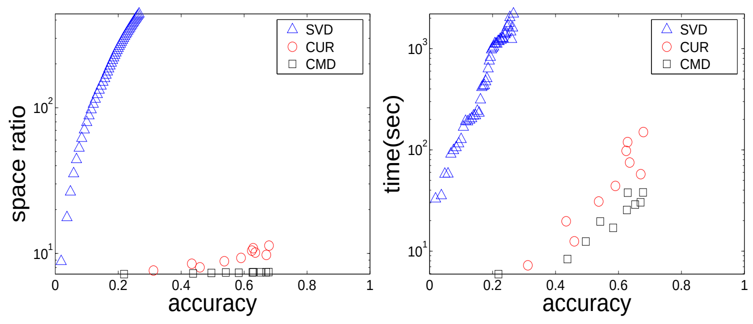
Figure 22: Network flow: CMD takes the least amount of space and time to decompose the source-destination matrix; the space and time required by CUR increases fast as the accuracy increases due to the duplicated columns and rows.

In terms of CPU time (see Figure 22), CMD achieves much more savings than SVD and CUR. There are two reasons: first, CMD does not need to process duplicate samples, and second, no expensive SVD is needed on the entire matrix (graph). CUR in general is faster to compute than SVD, but because of the duplicate samples, it spent a longer computation time than SVD to achieve a high accuracy. The majority of time spent by CUR is in performing SVD on the sampled columns.

**DBLP**: We observe similar performance trends using the DBLP dataset. CMD requires the least amount of space among the three methods (see Figure 23(a)). Notice that we do not show the high-accuracy points for SVD, because of its huge memory requirements. Overall, SVD uses more than 2000X more space than the original data, even with a low accuracy (less than $30\%$). The huge gap between SVD and the other two methods is mainly because: (1) the data distribution of DBLP is not as skewed as that of network flow, therefore the low-rank approximation of SVD needs more dimensions to reach the same accuracy, and (2) the dimension of left singular vector for DBLP (428,398) is much bigger than that for network flow

(21,837), which implies a much higher cost to store the result for DBLP than for network flow. These results demonstrates the importance of preserving sparsity in the result.

On the other hand, the difference between CUR and CMD in DBLP becomes less significant than that with network flow trace. The reason is that the data distribution is less skewed. There are fewer duplicate samples in CUR. In this case, CUR and CMD perform similarly.



(a) space vs accuracy (b) time vs. accuracy

Figure 23: DBLP: CMD uses the least amount of space and time. Notice the huge space and time that SVD requires.

The computational cost of SVD is much higher compared to CMD and CUR (see Figure 23). This is because the underlying matrix is denser and the dimension of each singular vector is bigger, which explains the high operation cost on the entire graph. CMD, again, has the best performance in CPU time for DBLP data.

## 4 Proposed Work

I propose to extend the current work in both *methodology* and *evaluation* aspects. The methodology map is listed in Table 3.

| | orthogonal projection | example-based | other divergence |
|---|---|---|---|
| Stream | [30, 33, 32, 21] | | |
| Matrix(Graph) | | [35] **[P1, P2]** | |
| Tensor | [34] | **[P3]** | **[P4]** |

Table 3: Methodology map: note that the solution to the bottom row is also a solution to the rows above.

### 4.1 P1: Effective example-based projection

CUR [13] and our CMD [35] provide a good start for summarizing matrices using an example-based projection instead of an orthogonal projection. Both of them give the following bound:

**Theorem 1 (CUR/CMD approximation)** *Given matrix* $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\|\mathbf{A} - \tilde{\mathbf{A}}\| \leq \|\mathbf{A} - \mathbf{A}_k\| + \epsilon\|\mathbf{A}\|$ *holds in expectation and with high probability*[10].

Note that this is a bound on the entire matrix norm $\|\mathbf{A}\|$ instead of the difference $\|\mathbf{A} - \mathbf{A}_k\|$. Ideally, we want a method with the relative bound, i.e., $\|\mathbf{A} - \tilde{\mathbf{A}}\| \leq (1+\epsilon)\|\mathbf{A} - \mathbf{A}_k\|$. There are theoretical advances along this line [14], but currently that method is even more expensive than SVD.

We plan to investigate the possibility to speed up the process by adopting some heuristics. For example, due to the sparsity of the data, the columns of the matrix are often in "near orthogonal" space. Namely, there are many columns that disagree in most of non-zero elements. For example, $(1,0,0,0)^T$ and $(0,1,1,0)^T$ disagree in all non-zero elements, which makes them orthogonal (i.e., the dot product of these two columns equals 0). In this case, a greedy approach can iteratively select a column that disagrees with most of selected columns (to span an orthogonal space) and has significant norm (to avoid numeric instability).

## 4.2 P2: Incremental CMD

Currently, we have completed the incremental model using orthogonal projection such as SPIRIT [30], distributed SPIRIT [33] and DTA/STA [34]. However, the example-based projection is not yet incremental, meaning that when a new matrix arrives, CMD has to run on that again without any reuse of historical computation.

We plan to study the possibility of the incremental CMD where CMD is computed or approximated incrementally over time. Possible problems include 1) how to reuse the past work? 2) when do we need to recompute from scratch? 3) do we ever need to recompute from scratch? 4) Can we provide any theoretical guarantee of the result?

One possible answer for the first question is to re-use the columns or rows sampled from previous time. The intuition is that the matrices (graphs) should be changing gradually if we assume there is a time dependency. For example, the network communication patterns in this hour are similar to the previous hour. Therefore, the subspace constructed from previous time may still be good for the current time. And there are many options on how to execute this re-use strategy. For instance, 1) *Exact reuse*: We use the exact old columns and rows from previous time; 2) *Partial reuse*: We pick the same column and row IDs from previous time but use the current values of those columns and rows to construct the new subspace; 3) *Multiple reuse*: Instead of using the columns and rows from a single timestamp, we can aggregate the information from multiple timestamps to form the new subspace.

## 4.3 P3: Example-based tensor decomposition

Both CUR and CMD are implemented for matrices only. However, many applications have high order data. In theory, the generalization to tensors is not hard as described by Drineas et al. [15].

However, there are still a lot of the practical concerns on prototyping the idea. More importantly, a necessary infrastructure has to be set up to implement this idea. For example, how to store a sparse tensor? How to efficiently access the elements in a tensor (both individual elements and sub-tensors)? Most of existing tensor prototypes [2] only deal with dense tensor and work with small datasets. To my knowledge, the only sparse tensor implementation is a preliminary Matlab package by Kolda et al. [4] which has an

[10] [13] proves it in both Forbenius norm and spectral norm.

efficient storage format but does not have many access methods. We have implemented DTA/STA on top of that package. And we plan to do the same for the example-based tensor decomposition.

### 4.4 P4: Other Divergence (distance function)

Most of matrix decompositions such as SVD, CUR and CMD assume Gaussian distribution with Euclidean distance, which may not be realistic for many applications. For example, network traffic are always nonnegative. However, SVD type of approach may have negative values in the result which are hard to interpret.

In general, we want to relax both the distribution assumption and the distance measure in our model. More specifically, we plan to use the Bregman divergence [6] as a general divergence measure for computing the difference between two tensors. Meanwhile, the distribution assumption is automatically generalized to the exponential families because of the existence of a bijection between the exponential family and the Bregman divergence as shown in [5]. For example, $L_2$ distance and KL-divergence correspond to Gaussian and Multinomial distribution, respectively.

The goal is to develop an optimization algorithm such as a gradient descent algorithm on tensors using the Bregman divergence.

### 4.5 Evaluation plan

We plan to continue using the real data and real application to evaluate our methods. In particular, we plan to study the following data:

- *Network flow data*: (Thanks to Prof. Hui Zhang and Dr. Yinglian Xie.) I plan to spend more time to understand the data and application in the network forensics. The goal is to develop useful tools to find interesting patterns and identify real anomalies for network forensics.

- *Machine monitoring application*: from the data center of PDL (Thanks to Prof Greg Ganger). We continue developing the automatic monitoring system, InteMon [21].

- *DBLP and IMDB*: The bibliographic data and movie database provide good examples of social networks. The task is to spot communities and identify abnormal individuals.

Other possible datasets we consider to study are :

- *fMRI data*: (Thanks to Prof Tom Mitchell.) fMRI data is in the format of $< (x, y, z), v >$ where $(x, y, z)$ specifies the 3D coordinates and $v$ is the value at the location. These data form 3rd order tensors naturally. The goal is to classify the data.

- *Financial data*: Transaction datasets. The goal is anomaly detection.

## 5 Conclusion

We study incremental pattern discovery on streaming applications where the data are arriving continuously in real-time. The contributions are the following:

- We proposed the *tensor stream* as a general dynamic data model for diverse applications. Under this model data streams and time-evolving graphs become the first and second order special cases.

- We developed the streaming algorithms (SPIRIT, distributed SPIRIT) for first order tensor streams. We evaluated the methods on real data and developed two prototyped systems (one on the environmental sensor network and one on the machine monitoring for a data center)
- We proposed two incremental methods for tensor streams (DTA/STA) which are evaluated on different datasets.
- We studied alternative methods for constructing low dimensional subspace for matrices using biased sampling. The proposed method, CMD, achieves better computational and storage efficiency than the traditional method.
- We propose to study the possibility of the incremental CMD and further extend CMD for tensor streams.
- We plan to study other distance measure such as Bregman divergence for handling more distribution as Exponential family. The model under Kullback-Leibler divergence (a special case of Bregman divergence) can deal with nonnegative data.
- We will evaluate our methods on real data and applications such as network forensics, machine monitoring and social networks.

Finally, let me end the proposal with the proposed schedule in Table 4.

| | |
|---|---|
| 1-3 months | P1: Effective example-based projection |
| 4-6 months | P2: Incremental CMD |
| 7-8 months | P3: Example-based tensor decomposition |
| 9-11 months | P4: Other distance measure |
| 6-12 months | Writing thesis |
| after 12 months | Defense |

Table 4: Time schedule

## References

[1] http://www.informatik.uni-trier.de/ ley/db/. pages 18

[2] http://www.models.kvl.dk/source/. pages 26

[3] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *PODS*, 2002. pages 10

[4] Brett W. Bader and Tamara G. Kolda. Algorithm xxx: Matlab tensor classes for fast algorithm prototyping. *ACM Transactions on Mathematical Software*, to apprear. pages 26

[5] A. Banerjee, S. Merugu, I. S. Dhillon, and J. Ghosh. Clustering with bregman divergences. *Journal of Machine Learning Research*, 6:1705–1749, Oct 2005. pages 27

[6] L. M. Bregman. The relaxation method to find the common point of convex sets and its applications to the solution of problems in convex programming. *USSR Computational Mathematics and Mathematical Physics*, 7:200–217, 1967. pages 6, 27

[7] 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. pages 10

[8] Graham Cormode, Mayur Datar, Piotr Indyk, and S. Muthukrishnan. Comparing data streams using hamming norms (how to zero in). *TKDE*, 15(3), 2003. pages 10

[9] Graham Cormode and S. Muthukrishnan. Space efficient mining of multigraph streams. In *PODS*, 2005. pages 11

[10] L. de Lathauwer. *Signal Processing Based on Multilinear Algebra*. PhD thesis, Katholieke, University of Leuven, Belgium, 1997. pages 3, 10

[11] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990. pages 7

[12] Chris Ding and Jieping Ye. Two-dimensional singular value decomposition (2dsvd) for 2d maps and images. In *SDM*, 2005. pages 10

[13] P. Drineas, R. Kannan, and M.W. Mahoney. Fast monte carlo algorithms for matrices iii: Computing a compressed approximate matrix decomposition. *SIAM Journal of Computing*, 2005. pages 6, 8, 21, 22, 25, 26

[14] P. Drineas, M. W. Mahoney, and S. (Muthu) Muthukrishnan. Polynomial time algorithm for column-row based relative error low-rank matrix approximation. In *manuscript*, 2005. pages 26

[15] Petros Drineas and Michael W. Mahoney. A randomized algorithm for a tensor-based generalization of the svd. *technical report*, 2005. pages 10, 26

[16] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM*, 1999. pages 11

[17] Phillip B. Gibbons and Yossi Matias. New sampling-based summary statistics for improving approximate query answers. In *SIGMOD*, 1998. pages 10

[18] Sudipto Guha, D. Gunopulos, and Nick Koudas. Correlating synchronous and asynchronous data streams. In *KDD*, 2003. pages 7

[19] 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. pages 10

[20] Xiaofei He, Deng Cai, and Partha Niyogi. Tensor subspace analysis. In *NIPS*, 2005. pages 10

[21] Evan Hoke, Jimeng Sun, and Christos Faloutsos. Intemon : Intelligent system monitoring on large clusters. In *Submitted to VLDB*, 2006. pages 5, 11, 14, 25, 27

[22] P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *FOCS*, 2000. pages 10

[23] I.T. Jolliffe. *Principal Component Analysis*. Springer, 2002. pages 7, 13

[24] A. Kapteyn, H. Neudecker, and T. Wansbeek. An approach to n-mode component analysis. *Psychometrika*, 51(2):269–275, 1986. pages 10

[25] Flip Korn, H. V. Jagadish, and Christos Faloutsos. Efficiently supporting ad hoc queries in large datasets of time sequences. In *SIGMOD*, pages 289–300, 1997. pages 7

[26] Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew Tomkins. Extracting large-scale knowledge bases from the web. In *VLDB*, 1999. pages 11

[27] Jurij Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: Densification laws, shrinking diameters and possible explanations. In *SIGKDD*, 2005. pages 11

[28] S. Muthukrishnan. *Data streams: algorithms and applications*, volume 1. Foundations and Trends. in Theoretical Computer Science, 2005. pages 10

[29] Christos H. Papadimitriou, Hisao Tamaki, Prabhakar Raghavan, and Santosh Vempala. Latent semantic indexing: A probabilistic analysis. *JOURNAL OF COMPUTER AND SYSTEM SCIENCES*, 61(2):217–235, 2000. pages 5, 7

[30] Spiros Papadimitriou, Jimeng Sun, and Christos Faloutsos. Streaming pattern discovery in multiple time-series. In *VLDB*, pages 697–708, 2005. pages 1, 5, 7, 11, 14, 25, 26

[31] 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. pages 10

[32] Jimeng Sun, Spiros Papadimitriou, and Christos Faloutsos. Online latent variable detection in sensor networks. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, 2005. pages 11, 14, 25

[33] Jimeng Sun, Spiros Papadimitriou, and Christos Faloutsos. Distributed pattern discovery in multiple streams. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, 2006. pages 1, 5, 11, 13, 14, 25, 26

[34] Jimeng Sun, Dacheng Tao, and Christos Faloutsos. Beyond streams and graphs: Dynamic tensor analysis. In *CMU technical report*, 2006. pages 1, 5, 11, 25, 26

[35] Jimeng Sun, Yinglian Xie, Hui Zhang, and Christos Faloutsos. Compact matrix decomposition for large graphs: Theory and practice. In *CMU technical report*, 2006. pages 1, 5, 11, 22, 25

[36] L. R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966. pages 10

[37] M. A. O. Vasilescu and D. Terzopoulos. Multilinear analysis of image ensembles: Tensorfaces. In *ECCV*, pages 447–460, 2002. pages 10

[38] N. Viereck, M. Dyrby, and S. B. Engelsen. *Monitoring Thermal Processes by NMR Technology*. Elsevier Academic Press, 2006. pages 10

[39] Jeffrey Scott Vitter. Random sampling with a reservoir. *ACM Trans. Math. Software*, 11(1):37–57, 1985. pages 10

[40] Dong Xu, Shuicheng Yan, Lei Zhang, Hong-Jiang Zhang, Zhengkai Liu, and Heung-Yeung Shum. Concurrent subspaces analysis. In *CVPR*, pages 203–208, 2005. pages 10

[41] Xifeng Yan, Philip S. Yu, and Jiawei Han. Substructure similarity search in graph databases. In *SIGMOD*, 2005. pages 11

[42] Jieping Ye. Generalized low rank approximations of matrices. *Machine Learning*, 61:167–191, 2004. pages 10