LWI-SVD: Low-rank, Windowed, Incremental Singular Value Decompositions on Time-Evolving Data Sets *

Xilun Chen, K. Selçuk Candan
Computer Science and Engineering School of Computing-IDSE (CIDSE), Arizona State University
Tempe, AZ, USA
xilun.chen@asu.edu, candan@asu.edu

ABSTRACT

Singular Value Decomposition (SVD) is computationally costly and therefore a naive implementation does not scale to the needs of scenarios where data evolves continuously. While there are various on-line analysis and incremental decomposition techniques, these may not accurately represent the data or may be slow for the needs of many applications. To address these challenges, in this paper, we propose a *Low-rank, Windowed, Incremental SVD* (LWI-SVD) algorithm, which (a) leverages efficient and accurate *low-rank approximations* to speed up incremental SVD updates and (b) uses a *window-based* approach to aggregate multiple incoming updates (insertions or deletions of rows and columns) and, thus, reduces online processing costs. We also present an *LWI-SVD with restarts* (LWI2-SVD) algorithm which leverages a novel highly efficient *partial reconstruction* based change detection scheme to support timely refreshing of the decomposition with significant changes in the data and prevent accumulation of errors over time. Experiment results, including comparisons to other state of the art techniques on different data sets and under different parameter settings, confirm that LWI-SVD and LWI2-SVD are both efficient and accurate in maintaining decompositions.

1. INTRODUCTION

Feature selection and dimensionality reduction techniques [20] usually involve some (often linear) transformation of the vector space containing the data to help focus on a few features (or combinations of features) that best discriminate the data in a given corpus. For example, the singular value decomposition (SVD [7]) of a data feature matrix A is of the form $A = USV^T$, where the r *orthogonal* column vectors of U form an r dimensional basis in which the n data objects can be described. Also, the r *orthogonal* column vectors of V (or the rows vector of V^T) form an r dimensional basis in which the m features can be placed. These r dimensions are referred to as the *latent variables* [16] or the *latent semantics* of the database [7]: Intuitively, the columns of U can be thought of as the eigen-objects of the data, each corresponding to one independent concept/cluster, and the columns of V can be thought of as the eigen-features of the collection, each, once again, corresponding to a concept/cluster in the database. In other words, SVD can be used for co-clustering both data-objects and features simultaneously. The $r \times r$ diagonal matrix S, can be considered to represent the strength of the corresponding latent concepts in the database: the amount of error caused by the removal of a concept from the database is proportional to the corresponding singular value.

*This work is partially funded by NSF grants #1339835 and #1318788. This work is also supported in part by the NSF I/UCRC Center for Embedded Systems established through the NSF grant #0856090.

KDD'14, August 24–27, 2014, New York, NY, USA.

http://dx.doi.org/10.1145/2623330.2623671.

1.1 Incremental SVD and Related Works

SVD is computationally costly and therefore a naive implementation does not match the real-time needs of scenarios where data evolve continuously: decomposition of an $n \times m$ matrix requires $O(n \times m \times min(n, m))$ time. While there are various on-line techniques, these are often slow or inaccurate. For example, one of the fastest techniques, SPIRIT [13] focuses on row insertions and cannot directly handle row deletions or column insertions/deletions. While a forgetting factor can be introduced to discount old objects, it cannot immediately reflect the properties of the removed entries on the decomposition. Moreover, since SPIRIT primarily considers data insertions and deletions, it is not applicable in situations where features of interest themselves evolve with the data (examples include weights of tags extracted from data and proximity to the hubs within an evolving network). As we see in Section 5, it has a higher inaccuracy compared to other incremental techniques, such as [5].

Other incremental SVD algorithms, such as [5, 6, 8, 9, 11, 12, 14, 15, 17], operate on an existing SV decomposition by folding-in new data and features into an existing (often low-rank) SVD; algebraic matrix manipulation techniques are used to rewrite the new SV decomposition matrices in terms of the old SV decomposition and update (including downdating) matrices. [5] showed that a number of database updates (including removal of columns) can all be cast as additive modifications to the original $n \times m$ database matrix, A. These updates then can be reflected on the SVD in $O(nmr)$ time as long as the rank, r, of the matrix A is such that $r \leq \sqrt{min(p, q)}$, where p is the number of new rows and q is the number of new columns. In other words, as long as the latent dimensionality of the database is low, the singular value decomposition can be updated in linear time. [5] further showed that the update to SVD can be computed in a single pass over the data matrix making the process highly efficient for large data. This and other existing algorithms can nevertheless be slow for many real-time applications.

1.2 Contributions of this Paper

In Section 2 we formalize these challenges and in Section 3, we propose a *Low-rank, Windowed, Incremental SVD* (LWI-SVD) algorithm, which leverages efficient and accurate *low-rank approxi-*

mations to speed up incremental SVD updates and uses a *window-based* approach to aggregate multiple incoming updates (insertion or deletions) and, thus reduces on-line costs. We also present, in Section 4, an LWI2-SVD algorithm which leverages a novel partial reconstruction based change detection technique to support timely refreshing of the decompositions to prevent accumulation of errors.

Experiment results reported in Section 5 confirm that LWI-SVD and LWI2-SVD are both efficient and accurate in maintaining decompositions. We conclude the paper in Section 6.

2. BACKGROUND

2.1 Problem Definition

At time stamp i, we are given a set of n data tuples $D_i = \{t_{i,1}, t_{i,2}, ..., t_{i,n}\}$, each with a set, F_i, of m features. We are also given the set, L_i, containing the r latent semantics of D_i (and their weights). As time moves, new tuples arrive and some of the existing tuples expire: at the next time stamp, t_{i+1}, the tuple set is $D_{i+1} = (D_i \backslash \Delta D_{i+1}^-) \cup \Delta D_{i+1}^+$, where ΔD_{i+1}^- are the tuples that expired and ΔD_{i+1}^+ are the new tuples that arrived. Moreover, at time $(i+1)$, we have a new set, F_{i+1}, of features, where $F_{i+1} = (F_i \backslash \Delta F_{i+1}^-) \cup \Delta F_{i+1}^+$, where ΔF_{i+1}^- are features that are not of interest anymore and ΔF_{i+1}^+ are the new features of interest.

Our goal is to quickly obtain L_{i+1} containing the r latent semantics corresponding to time instance, $i+1$, and efficiently maintain these r latent semantics as time further moves.

2.2 Basic Incremental SVD [5]

Let us be given an $n \times m$ data matrix $X = U_x S_x V_x^T$ and an $n' \times m'$ updated data matrix $X' = X + \Delta$, where Δ is a $max(n,n') \times max(m,m')$ change matrix. Note that if X' has larger dimension than X, X is padded with $n'-n$ rows of zero and $m'-m$ columns of zero to match the dimension of Δ, the removal of rows and columns are modeled by additions that result in zeroing of the corresponding rows and columns (which are then dropped from the matrix). Let us further assume that the change matrix Δ can be decomposed into $\Delta = AB^T$. Note that we can rewrite the matrix X' as

$$X' = X + AB^T = \begin{bmatrix} U_x & A \end{bmatrix} \begin{bmatrix} S_x & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} V_x & B \end{bmatrix}^T. \quad (1)$$

Given these, [5] incrementally maintains SVD as follows:

2.2.1 QR Decompositions

Let us also define Q_A as the orthogonal basis of $(I - U_x U_x^T)A$ and Q_B as the orthogonal basis of $(I - V_x V_x^T)B$. Both Q_A and Q_B can be obtained through QR decomposition [4] of $(I - U_x U_x^T)A$ and $(I - V_x V_x^T)B$:

$$Q_A R_A = (I - U_x U_x^T)A; \qquad Q_B R_B = (I - V_x V_x^T)B \quad (2)$$

Here Q_A and Q_B are orthogonal matrices and R_A and R_B are upper-triangular. It is easy to see, through basic matrix algebra, that the following holds:

$$\begin{bmatrix} U_x & A \end{bmatrix} = \begin{bmatrix} U_x & Q_A \end{bmatrix} \begin{bmatrix} I & U_x^T A \\ 0 & R_A \end{bmatrix} \quad (3)$$

$$\begin{bmatrix} V_x & B \end{bmatrix} = \begin{bmatrix} V_x & Q_B \end{bmatrix} \begin{bmatrix} I & V_x^T B \\ 0 & R_B \end{bmatrix} \quad (4)$$

Moreover, by substituting Equations 3 and 4 into Equation 1, we can get

$$X' = X + AB^T = \begin{bmatrix} U_x & Q_A \end{bmatrix} K \begin{bmatrix} V_x & Q_B \end{bmatrix}^T \quad (5)$$

where K is equal to

$$K = \begin{bmatrix} I & U_x^T A \\ 0 & R_A \end{bmatrix} \begin{bmatrix} S_x & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} I & V_x^T B \\ 0 & R_B \end{bmatrix}^T \quad (6)$$

$$= \begin{bmatrix} S_x & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} U_x^T A \\ R_A \end{bmatrix} \begin{bmatrix} V_x^T B \\ R_B \end{bmatrix}^T \quad (7)$$

2.2.2 Matrix K

Let us remember that X is an $n \times m$ matrix and X' is an $n' \times m'$ matrix. Given this

- if $n \geq n'$ and $m \geq m'$, K is a matrix of size $(n+1) \times (m+1)$. This is because, if $n \geq n'$, then U_x is an orthogonal matrix and $U_x U_x^T$ is equal to I. Consequently, $Q_A R_A = (I - U_x U_x^T)A = 0$ and this implies that R_A is simply 0. The same is true for R_B.
- if $n < n'$ and $m < m'$, K is a matrix of size $n' \times m'$. In Section 3.2.1, we discuss the shape K takes in this case and the resulting properties in detail.
- if $n \geq n'$ and $m < m'$, K is a matrix of size $(n+1) \times m'$.
- if $n < n'$ and $m \geq m'$, K is a matrix of size $n' \times (m+1)$.

2.2.3 Using the Decomposition of K to Obtain the Decomposition of X'

Let us consider the SV decomposition of K; i.e., $K = U_K S_K V_K^T$. Equation 1 can be rewritten [5] as

$$X' = X + AB^T = (\begin{bmatrix} U_x & Q_A \end{bmatrix} U_K) S_K (\begin{bmatrix} V_x & Q_B \end{bmatrix} V_K)^T \quad (8)$$

giving us the SVD of the new tuple matrix, X'.

The challenge, of course, is to obtain the matrices, Q_A and Q_B, and the SV decomposition of K efficiently. In order to keep the complexity down, [5] suggests that A and B should be taken as combination of simple column vectors so that AB^T can be the sum of multiple rank-1 matrices. This, however, may be a significant constraint in real-applications where the change matrix Δ itself can have a large size, indicating great amount of rank-1 matrices it produces and updating a sequence of rank-1 matrix is not effective as treating them as a whole. In the next section, we discuss how to relax this assumption of [5] without impacting efficiency and accuracy.

3. LWI-SVD

We now present our key ideas for efficient incremental SVD operations. As described above, this involves efficiently searching for matrices, Q_A and Q_B, and the SVD of K.

3.1 Efficiently Obtaining Q_A and Q_B

As described above, Q_A is the orthogonal basis of $(I - U_x U_x^T)A$ and Q_B is the orthogonal basis of $(I - V_x V_x^T)B$. These can be obtained using two expensive QR decomposition operations for both Q_A and Q_B. One way to reduce the number of QR decomposition operations would be to seek a decomposition of Δ where $X' = X + \Delta = AA^T$; i.e., $A = B$. However, not all Δ will have such a convenient decomposition. When Δ is negative definite, it cannot be written as the format of $A \times B$ where $A = B$.

Instead, in this paper, we propose to reduce the cost of the overall QR decomposition step by setting A to the identity matrix I and setting B^T to Δ. This does not lose any generality on the algorithn since Δ (B^T) can be any matrix. When we do this, since $A = I$, it would also be the case that $Q_A = \begin{bmatrix} 0 \\ I \end{bmatrix}$. Therefore, we need only one QR decomposition. What is more, if the Δ only reflect a small amount of data insertions and deletions, then it will be a sparse matrix with last few rows and columns of nonzero values. This

lead to efficient computation of $(I - V_x V_x^T)B$ and $V_x^T B$ by block matrix multiplication. Let's first find the zero block of B when it is data insertion. Then Δ (B) is a $n' \times m'$ matrix with a block of zero values on the first $n \times m$ position. We can rewrite B as

$$B = \begin{bmatrix} 0 & B_1 \\ B_2 & B_3 \end{bmatrix}$$

Then, we can divide $(I - V_x V_x^T)$ and V_x^T into the same block size as B. For example, we can rewrite V_x^T as

$$V_x^T = \begin{bmatrix} {V_x}_0^T & {V_x}_1^T \\ {V_x}_2^T & {V_x}_3^T \end{bmatrix}$$

Then, the multiplication of $V_x^T B$ becomes

$$V_x^T \times B = \begin{bmatrix} {V_x}_1^T \times B_2 & {V_x}_0^T \times B_1 + {V_x}_1^T \times B_3 \\ {V_x}_3^T \times B_2 & {V_x}_2^T \times B_1 + {V_x}_3^T \times B_3 \end{bmatrix}$$

Note that, the multiplication of ${V_x}_0^T$ and the corresponding block of B is avoided since the corresponding block of B is all zeros. Also, the other part of V_x^T and B are small size thin matrices. Thus, the multiplication of $V_x^T \times B$ can be done very efficiently. The same applies to $(I - V_x V_x^T) \times B$ and when the data are deleted.

As we experimentally show in Section 5.4.1, this optimization provides significant gains in time, without any noticeable loss in the final accuracy.

3.2 Efficiently Decomposing K

The next challenge is to obtain the singular value decomposition of the matrix, K. Performing SVD on K directly would be costly as the SVD operation is expensive. However, as we prove next, in Section 3.2.1, in the presence of row and column insertions, K takes a special structure:

$$K = \begin{bmatrix} S_x & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} U_x^T A \\ R_A \end{bmatrix} \begin{bmatrix} V_x^T B \\ R_B \end{bmatrix}^T = \begin{bmatrix} S_x & \Pi \\ \Phi & \Gamma \end{bmatrix}.$$

More specifically, in the presence of insertions, (a) since S_x is diagonal, K is mostly sparse, and (b) it is shaped like an arrow: (aside from the diagonal) there are non-zeros only on its last rows and columns. We verify these next.

3.2.1 Shape of K

Let X be an $n \times m$ matrix and X' be an $n' \times m'$ matrix. In Section 2.2.2, we have seen that K is either of size $n' \times m'$, $(n+1) \times (m+1)$, $(n+1) \times m$, or $n' \times (m+1)$, depending on whether the numbers of rows and columns increase or decrease when the data matrix transforms from X to X'. Let us further assume that $n \leq m$ and $m' > m$ and $n' > n$, which is rows and columns insertion. As we already discussed before, let us set $A = I_{n'}$ and $B^T = \Delta$, where $A \in \mathbb{R}^{n' \times n'}$, $B \in \mathbb{R}^{m' \times n'}$ and $\Delta \in \mathbb{R}^{n' \times m'}$ so that AB^T is equal to the update matrix Δ. Finally, let SVD of X be $X = U_x S_x V_x^T$, or simply $X = USV^T$.

Given the fact that $X' = X + \Delta$, we can also deduce that $X' = U'SV'^T + \Delta$, where

$$U' = \begin{bmatrix} U \\ 0 \end{bmatrix} \in \mathbb{R}^{n' \times n} \text{ and } V' = \begin{bmatrix} V \\ 0 \end{bmatrix} \in \mathbb{R}^{m' \times m}.$$

Intuitively, U and V are augmented by padding $n' - n$ rows of zeros to U and $m' - m$ rows of zeros to V to make it compatible with Δ. This padding gives us

$$X' = \begin{bmatrix} X & 0 \\ 0 & 0 \end{bmatrix} + \Delta = U'SV'^T + \Delta.$$

Secondly, using a similar zero-padding, we can get the following equalities:

$$(I_{n'} - U'U'^T)A = \begin{bmatrix} 0 & 0 \\ 0 & I_{n'-n} \end{bmatrix} \quad (9)$$

$$(I_{m'} - V'V'^T)B = \begin{bmatrix} 0 \\ B_{m'-m} \end{bmatrix} \quad (10)$$

The right hand side of Equation 9 has $n' - n$ independent columns and, thus, it has a simple QR decomposition:

$$Q_A = \begin{bmatrix} 0 \\ I_{n'-n} \end{bmatrix} \text{ and } R_A = \begin{bmatrix} 0 & I_{n'-n} \end{bmatrix}$$

Since the right hand side of the Equation 10 consists of 0s except for the last $m' - m$ rows, the QR decomposition of the left hand side will be such that $Q_B \in \mathbb{R}^{(m'-m) \times m}$ and $R_B \in \mathbb{R}^{(m'-m) \times n'}$. Let us further partition R_B into two,

$$R_B = \begin{bmatrix} R_{B1} & R_{B2} \end{bmatrix},$$

where $R_{B1} \in \mathbb{R}^{(m'-m) \times n}$ and $R_{B2} \in \mathbb{R}^{(m'-m) \times (n'-n)}$.

Given the above, we can rewrite the matrix, K, as

$$K = \begin{bmatrix} S & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} U'^T A \\ R_A \end{bmatrix} \begin{bmatrix} V'^T B \\ R_B \end{bmatrix}^T$$

where

$$\begin{bmatrix} U'^T A \\ R_A \end{bmatrix} = \begin{bmatrix} U^T & 0 \\ 0 & I_{n'-n} \end{bmatrix}$$

$$\begin{bmatrix} V'^T B \\ R_B \end{bmatrix}^T = \begin{bmatrix} 0 & Y \\ R_{B1} & R_{B2} \end{bmatrix}^T.$$

Here Y is a $m \times (n' - n)$ matrix. Note that we can further rewrite

$$\begin{bmatrix} U'^T A \\ R_A \end{bmatrix} \begin{bmatrix} V'^T B \\ R_B \end{bmatrix}^T = \begin{bmatrix} U'^T A \\ R_A \end{bmatrix} \begin{bmatrix} (V'^T B) \\ R_{B1} & R_{B2} \end{bmatrix}^T$$

as

$$\begin{bmatrix} U^T & 0 \\ 0 & I_{n'-n} \end{bmatrix} \begin{bmatrix} 0 & Y \\ R_{B1} & R_{B2} \end{bmatrix}^T = \begin{bmatrix} 0 & U^T R_{B1}^T \\ Y^T & R_{B2}^T \end{bmatrix}.$$

Thus, K simplifies to

$$K = \begin{bmatrix} S & U^T R_{B1}^T \\ Y^T & R_{B2}^T \end{bmatrix} = \begin{bmatrix} S & \Pi \\ \Phi & \Gamma \end{bmatrix}.$$

This confirms that when $m < m'$ and $n < n'$, K is shaped like an arrow: it is diagonal, except for the last $n' - n$ rows and last $m' - m$ columns. This, however, is not true when $m \geq m'$ or $n \geq n'$; in this case K can be a dense matrix, with its last row and columns equal to 0. In the rest of this section, we argue that, especially when $m < m'$ and $n < n'$, we can leverage K's specific structure (sparse, arrow-like) to quickly obtain a highly-accurate approximate decomposition, $K \sim \hat{U}\hat{S}\hat{V}^T$ and use it instead of the exact decomposition $K = U'S'V'^T$. In particular we propose to build on the *SVD through QR decomposition with column pivoting* technique proposed in [4]. Experiment results reported in Section 5 show that this leads to efficient and accurate decompositions even in cases where $m \geq m'$ or $n \geq n'$.

3.2.2 Decomposition of K through Pivoted QR

Pivoted QR Factorization. Let E be a matrix. A pivoted QR factorization of E has the form $EP = Q_e R_e$ where P is a permutation matrix, Q_e is orthonormal and R_e is upper triangular. [4] has shown that a rank-k approximation can be obtained efficiently through a pivoting process where columns of E are considered one at a time and used to compute an additional column of Q_e and

row of R_e. The k^{th} round of the process leads to a rank-k approximation of the pivoted QR factorization of E. In particular, let us assume that we are given a QR decomposition of the form $F = Q_f R_f$ and need to compute QR decomposition of $[F\ a]$ for some column vector a:

$$[F\ a] = [Q_f\ q] \begin{bmatrix} R_f & \epsilon \\ 0 & \rho \end{bmatrix}$$

The rank-k approximation can be obtained efficiently by the quasi-Gram-Schmidt method, which further eliminates the need to store dense Q_f matrices [4]: the quasi-Gram-Schmidt process can be applied successively to columns of a given input matrix E to produce a pivoted QR factorization for E.

Low-Rank Decomposition of K. Let us assume that we are targeting a rank-k decomposition of K. We first sample k columns to obtain column-sample matrix $\mathcal{C}$; we then sample k columns from K^T to obtain a row-sample matrix $\mathcal{R}^T$. We then apply the QR decomposition with column pivoting to $\mathcal{C}$ and $\mathcal{R}^T$ to obtain upper triangular matrices, R_c and R_r.

The sampling is done by selecting the longest row and column vectors. We note that when $m < m'$ and $n < n'$, K is not only sparse, but also has an arrow-like shape:

$$K = \begin{bmatrix} S_x & \Pi \\ \Phi & \Gamma \end{bmatrix},$$

where the $n \times m$ matrix S_x is diagonal, whereas $n \times (m' - m)$ matrix Π, $(n' - n) \times m$ matrix Φ, and $(n' - n) \times (m' - m)$ matrix Γ are potentially dense as we discussed in Section 3.2.1. As a result, the sampling is *arrow-sensitive* in the sense that it focuses on the last few rows and columns: The sampled columns usually come from the first few columns (which contain the largest singular values at the top-left corner of the matrix) and the last few columns, which contain entries from the dense, $\begin{bmatrix} \Pi \\ \Gamma \end{bmatrix}$. Similarly, the sampled rows come from the first few rows (which contain large singular values in S_x) and the last few rows from $\begin{bmatrix} \Phi & \Gamma \end{bmatrix}$.

Given these, to obtain a decomposition of K, we need to find a matrix H such that $\|K - \mathcal{C}H\mathcal{R}^T\|$ is minimized. According to [15], the value of H which minimizes this can be computed as

$$(R_c^{-1}R_c^{-T})(\mathcal{C}^T K \mathcal{R})(R_r^{-1}R_r^{-T}).$$

Thus, we can rewrite $\mathcal{C}H\mathcal{R}^T$ as

$$(\mathcal{C}R_c^{-1})(R_c^{-T}\mathcal{C}^T K \mathcal{R} R_r^{-1})(R_r^{-T}\mathcal{R}^T).$$

If we further set $W = R_c^{-T}\mathcal{C}^T K \mathcal{R} R_r^{-1}$ and decompose W into $W = U_w S_w V_w^T$, then we can obtain the SV decomposition of K as $K = U_K S_K, V_K^T$, where

$$U_K = \mathcal{C}R_c^{-1}U_w, \quad V_K^T = V_w^T R_r^{-1} R_r^{-T}, \text{ and } S_K = S_w,$$

where U_K and V_K are orthonormal and S_K is diagonal. While this process also involves an SV decomposition step involving W, since W is a much smaller, $k \times k$, matrix, its decomposition is much faster than the direct decomposition of K.

3.3 Pseudocode of LWI-SVD

Algorithm 1 provides the pseudo-code of the proposed Low-rank, Windowed, Incremental Singular Value Decomposition (LWI-SVD) algorithm for incrementally maintaining the SVD of an evolving matrix X. As we later see in Section 5, the LWI-SVD algorithm has a smaller approximation error than other algorithms, such as SPIRIT [13], yet is also much faster than optimal as well as the basic incremental SVD [5] algorithms. Yet, as in any incremental approximate algorithm, in which each step takes the output of the previous step as its input, there is a likelihood that errors will accumulate over time and the reconstruction error relative to the actual matrix will reach an unacceptable rate. To prevent errors to accumulate, in the next section we propose a novel *LWI-SVD with Restart* (LWI2-SVD) algorithm which restarts the SVD by performing a fresh SVD on the current data matrix.

Algorithm 1 LWI-SVD.

Input:
The Base Matrix, X, and its SV decomposition $U_x S_x V_x^T$;
The update matrix, $\Delta = AB^T$, corresponding to a window of updates;
Target rank, r;

Output:
The new SVD results, U'_x,S'_x, and V'_x;

1: Calculate factors R_A and R_B in Equation 2 which, as discussed in Section 3.1, involves a QR Decomposition and several matrix multiplications;
2: Calculate the matrix K in Equation 7;
3: Obtain the low-rank (rank-r) decomposition of K into $K = U_K S_K V_K^T$;
4: Combine the factors as shown in Equation 8 to obtain rank-r decomposition U'_x, S'_x, and V'_x;
5: **return** U'_x, S'_x, and V'_x;

4. LWI2-SVD: LWI-SVD WITH RESTART

In this section, we build on LWI-SVD and propose a novel *LWI-SVD with Restart* (LWI2-SVD) algorithm which punctuates the incremental SVD sequence by occasionally performing a full SVD on the current data matrix. Obviously, there is a direct, positive correlation between the frequency of restarts and the overall accuracy of the LWI2-SVD algorithm. Unfortunately, however, there is also a strong positive correlation between the cost of LWI2-SVD and the frequency of restarts. Therefore, restart rate should be such that the process is restarted only when the costly SVD is in fact needed to help reduce the overall error.

4.1 Types of Errors

We see that there are two distinct types of errors:

- *Accumulated approximation errors (and periodic restarts):* The first type of error that accumulates over time is due to the various approximation terms, including the low-rank approximation of K as discussed in Section 3.2. While the absolute value of this error will be different from one iteration of the algorithm to the next, its long term behavior will be roughly constant. Therefore, this type of accumulated approximation errors are best dealt with *periodic restarts*.
- *Error bursts due to structural changes in the input data (and on-demand restarts):* The second type of error in the incremental SVD occurs when there is a significant structural (or spectral) change in the data, necessitating large changes in the SVD. Since the incremental process described in Section 3 assumes that the changes are relatively small, a significant structural change in the factor matrices, U_x and V_x, or the core matrix S_x may not be correctly captured, resulting in a large burst of reconstruction error. These bursts are best dealt with *on-demand restarts* that are triggered through a change detection process that tracks the updates to identify when major structural changes in the data occur.

Figure 1 shows an example run with and without restarts. Note that without the restarts errors continuously accumulate due to structural changes in the data. Restarts (both periodic and on-demand)

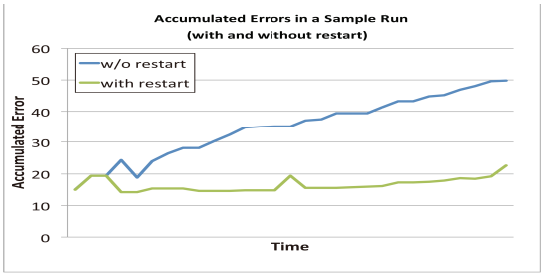


Figure 1: Example runs with and without restarts

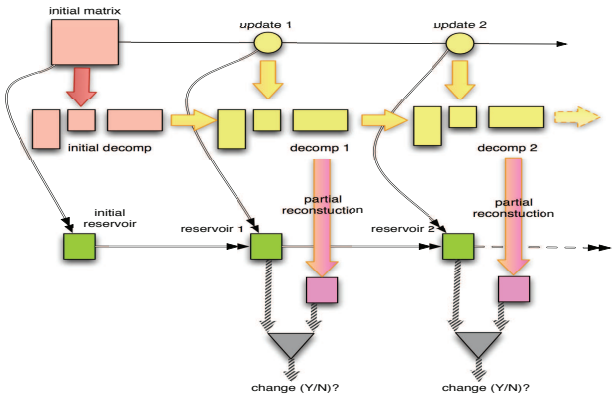


Figure 2: Overview of the change detection process

can limit the accumulation of errors. Error accumulations due to approximations generally show a regular behavior and the frequency with which periodic restarts are scheduled can be set empirically. The structural changes in the data, however, do not necessarily have a regular behavior; therefore, the challenge is to quickly and efficiently detect the structural changes in the data. We will discuss this next.

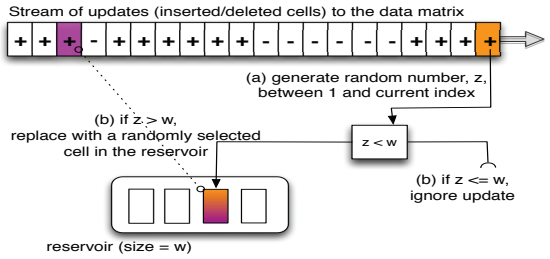
4.2 Change Detection through Partial Reconstruction

In order to detect major structural changes in the data we need to measure or estimate the reconstruction errors. The naive way to achieve this would be to reconstruct the entire matrix from the incrementally maintained decomposition and compare the reconstructed matrix to the ground truth (which is the actual, revised data matrix). If the difference is high, it means that due to some structural changes, the incrementally maintained decomposition deviated from the true decomposition of the matrix. Obviously, performing a full reconstruction of the matrix at each time step would be extremely costly. Instead, in this section, we propose a change detection scheme which relies on a partial reconstruction as depicted in Figure 2: (a) a fair data matrix sampler, which identifies a small subset of the matrix cells as ground truth and (b) a partial reconstructor, which reconstructs a given subset of matrix cells, without reconstructing the full data matrix.

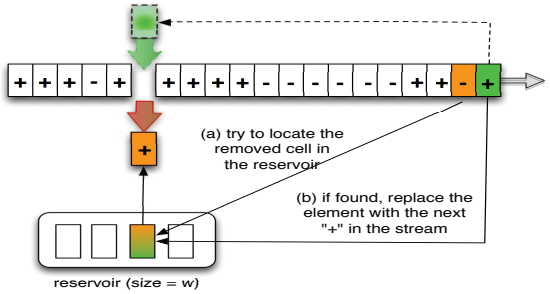
4.2.1 Fair Sampling of an Evolving Matrix

We propose a fair sampler, where all matrix cells have a uniform probability of being selected independently of when they are updated.

Basic Reservoir Sampling. Reservoir sampling [18] is a random sampling method that works well in characterizing data streams. It is especially efficient because (a) it needs to see the data only once



(a) reservoir maintenance for $\mathbf{s_i} = +\langle \mathbf{row_i}, \mathbf{col_i} \rangle$



(b) reservoir maintenance for $\mathbf{s_i} = -\langle \mathbf{row_i}, \mathbf{col_i} \rangle$

Figure 3: Overview of the reservoir based matrix sampling

and (b) it uses a fixed (and small) buffer, referred to as the "*reservoir*". Furthermore, while (c) it does not require a priori knowledge of the data size, it (d) ensures that each data element has an equal chance of being represented in the sample. Let $\mathcal{S}$ be a data stream consisting of a sequence of elements s_i. The reservoir sample keeps a fixed reservoir of, say w elements. Once the reservoir is full, each new element, s_i, replaces a (randomly) selected element in the reservoir with a decreasing probability, inversely proportional to the index, i, of the new element s_i. More specifically, a random element in the reservoir is replaced by s_i with probability $\frac{w}{i}$. Intuitively, in a fair sampling, each element up to i should have a w/i chance of being in the random sample of size w. Therefore, s_i is selected to be included in the reservoir with probability $\frac{w}{i}$. The sample it replaces, on the other hand, is chosen randomly among the existing w samples in the reservoir to ensure that the reservoir forms a random sample of the first i elements in the stream.

Matrix-Reservoir Model. As we described earlier, we consider the general case where the data matrix can grow or shrink with insertions or deletions of rows and columns. More specifically, we model the evolving data matrix as a stream, $\mathcal{S}$, of $s_i = \pm\langle row_i, col_i \rangle$, where row_i and col_i are the row and columns affected in the update with index i: $+\langle row_i, col_i \rangle$ indicates that the update inserts a new cell in the matrix at location $\langle row_i, col_i \rangle$, whereas $-\langle row_i, col_i \rangle$ indicates that the cell at location $\langle row_i, col_i \rangle$ is being removed.

The reservoir, $\mathfrak{R}_i = \{\mathfrak{r}_{i,1}, \ldots, \mathfrak{r}_{i,w}\}$, at time i consists of w matrix cell positions, which serve as the representatives for the current matrix. In other words, each $\mathfrak{r}_{i,j} \in \mathfrak{R}_i$ is a triple of the form $\mathfrak{r}_{i,j} = \langle index_{i,j}, row_{i,j}, col_{i,j} \rangle$, where $index_{i,j}$ is the index of the update that deposited the cell, located at $row_{i,j}$ and $col_{i,j}$, into the reservoir.

Matrix-Reservoir Maintenance for $\mathbf{s_i} = +\langle \mathbf{row_i}, \mathbf{col_i} \rangle$. As discussed earlier, reservoir sampling randomly selects some of the incoming stream elements for the updating the contents of the reservoir When the (probabilistically) selected incoming stream entry s_i is of the form $+\langle row_i, col_i \rangle$, the basic reservoir sampling process is applied: a random element, $\mathfrak{r}_{i-1,j}$ from the current reservoir

$\mathfrak{R}_{i-1}$ is selected and this is replaced with $\langle i, row_i, col_i \rangle$. This process is visualized in Figure 3(a).

Matrix-Reservoir Maintenance for $\mathbf{s_i} = -\langle \mathbf{row_i}, \mathbf{col_i} \rangle$. When the (probabilistically) selected incoming entry s_i is of the form $-\langle row_i, col_i \rangle$, on the other hand, the basic reservoir sampling process cannot be applied as this denotes removal of a cell, not insertion. We handle deletions as follows:

- if there exists no
  $$\mathfrak{r}_{i-1,j} = \langle index_{i-1,j}, row_{i-1,j}, col_{i-1,j} \rangle \in \mathfrak{R}_{i-1},$$
  such that $row_{i-1,j} = row_i$ and $col_{i-1,j} = col_i$, then s_i is simply ignored;
- if, on the other hand, there exists a
  $$\mathfrak{r}_{i-1,j} = \langle index_{i-1,j}, row_{i-1,j}, col_{i-1,j} \rangle \in \mathfrak{R}_{i-1},$$
  such that $row_{i-1,j} = row_i$ and $col_{i-1,j} = col_i$, then
  - we drop $\mathfrak{r}_{i-1,j}$ from the reservoir and
  - we keep the j^{th} position reserved for a future update of the form $s_h = +\langle row_h, col_h \rangle$.

Intuitively, the matrix reservoir (and its history) is revised as if the future insertion s_h had in fact arrived *in the past*, instead of $s_{index_{i-1,j}}$, which had originally deposited the cell, $\langle row_{i-1,j}, col_{i-1,j} \rangle$ (which is being deleted) into the reservoir. This process is visualized in Figure 3(b).

4.2.2 Partial Matrix Reconstruction

At time $t = i$, let us have the reservoir $\mathfrak{R}_i = \{\mathfrak{r}_{i,1}, \ldots, \mathfrak{r}_{i,w}\}$, where for all $1 \leq h \leq w$, $\mathfrak{r}_{i,h} = \langle index_{i,h}, row_{i,h}, col_{i,h} \rangle$. Intuitively, the reservoir consists of a set of matrix cell positions (that were fairly sampled from the overall matrix). During the partial reconstruction step, we use the (incrementally maintained) SV decomposition, U_i, S_i, and V_i, of the data matrix X_i to reconstruct only the row and column positions that appear in the reservoir, $\mathfrak{r}_{i,h}$.

More formally, the partially reconstructed matrix value set $\hat{\mathfrak{V}}_i = \{\hat{\mathfrak{v}}_{i,1}, \ldots, \hat{\mathfrak{v}}_{i,w}\}$, is such that for all $1 \leq h \leq w$,

$\hat{\mathfrak{v}}_{i,h} = \hat{X}_i[row_{i,h}, col_{i,h}]$, where

$$\hat{X}_i[row_{i,h}, col_{i,h}] = (U_i[row_{i,h}, *])\, S_i \left(V_i^T[*, col_{i,h}]\right).$$

Note that the cost of the partial reconstruction of the matrix depends on the size of the reservoir and when $|\mathfrak{R}_i| \ll |X_i|$, partial reconstruction is much faster than full reconstruction.

4.2.3 Change Detector

At time $t = i$, given the reservoir $\mathfrak{R}_i = \{\mathfrak{r}_{i,1}, \ldots, \mathfrak{r}_{i,w}\}$, we construct a ground truth value set $\mathfrak{V}_i = \{\mathfrak{v}_{i,1}, \ldots, \mathfrak{v}_{i,w}\}$, where for all $1 \leq h \leq w$, $\mathfrak{v}_{i,h} = X_i[row_{i,h}, col_{i,h}]$. Similarly, we also have the partially reconstructed value set $\hat{\mathfrak{V}}_i = \{\hat{\mathfrak{v}}_{i,1}, \ldots, \hat{\mathfrak{v}}_{i,w}\}$, where for all $1 \leq h \leq w$, $\hat{\mathfrak{v}}_{i,h} = \hat{X}_i[row_{i,h}, col_{i,h}]$, where $\hat{X}_i[row_{i,h}, col_{i,h}]$ is the partially reconstructed value for the cell location $\langle row_{i,h}, col_{i,h} \rangle$. Given these, we detect a major structural change in the data matrix if

$$\sum_{h=1}^{w} (\mathfrak{v}_{i,h} - \hat{\mathfrak{v}}_{i,h})^2 \geq \Theta,$$

where Θ is the inaccuracy threshold.

4.3 Pseudocode of the LWI2-SVD Algorithm

We provide the pseudocode of the LWI-SVD with Restart (LWI2-SVD), which was detailed in this section, in Algorithm 2. In the next section, we evaluate the efficiency and effectiveness gains of LWI2-SVD algorithm on top of the gains provided by LWI-SVD.

Algorithm 2 LWI2-SVD

Input:
The Base Matrix, X, and its SV decomposition $U_x S_x V_x^T$;
The update matrix, $\Delta = AB^T$, corresponding to a window of updates;
Target rank, r;
Reservoir, $\mathfrak{R}$;
Restart Threshold, Θ;
Periodic Restart Flag, f;

Output:
The new SVD results, U'_x,S'_x, and V'_x;
The new Reservoir, $\mathfrak{R}'$;

```
X' = X + Δ
if f = true then
    ⟨U'_x, S'_x, V'_x⟩ = topK_SVD(X', r);
    ℜ' = updateReservoir(ℜ, Δ);
else
    ⟨U'_x, S'_x, V'_x⟩ = LWI-SVD(X, U_x, S_x, V_x, Δ, r);
    ℜ' = updateReservoir(ℜ, Δ);
    𝔙̂ = partialReconstruct(ℜ', U'_x, S'_x V'_x);
    E = measurePartialError(𝔙̂, ℜ', X');
    if E > Θ then
        ⟨U'_x, S'_x, V'_x⟩ = topK_SVD(X', r);
    end if
end if
return U'_x, S'_x, V'_x, ℜ';
```

Table 1: Parameters

Symbol	Desc.	Default	Alternative
$dim(n \times n)$	Initial(for insertions)/Final(for deletions) dimensions of X	100×100	300×300
r	Target rank	5	10
len	Length of the data stream	50	50
num_{upd}	Numbers of columns:rows updated at a given iteration	2:2	6:6
λ_{upd}	Strength of the updates (for synth. data)	5	10
w	Reservoir size	50	150
Θ	On-demand restart threshold	20%	10%
per	Restart period	15	5

5. EXPERIMENTS

In this section, we evaluate the efficiency and effectiveness of LWI-SVD and LWI2-SVD on both synthetic and real datasets and for different scenarios and parameter settings.

Each experiment, consisting of len consecutive update iterations, was run 10 times and averages are reported. Note that to simplify the interpretation of the results we have considered *insertion sequences* and *deletion sequences*; but not hybrid *insertion/deletion* sequences. Also, to make sure that the results for experiments involving sequences of insertions and deletions are comparable, we have set the initial dimensions for an insertion sequence and the final dimensions of a deletion sequence to the same value, dim.

The various parameters varied in the experiments, default values, and value ranges are presented in Table 1. Below we describe the experimental setting, including the data sets, in greater detail.

5.1 Real Data: Digg.com Traces

We use Digg.com data set [2] from Infochimps to evaluate the effectiveness and efficiency for real data. The complete data set

was recorded from August to November 2008 and has 3 main components: stories, comments and replies. "Stories" contain 1490 articles that users have posted within the time period. For our experiments, we created data streams by considering the first $n + len \times num_{upd}$ articles in the data set(the first n articles make up the initial data matrix; for each of the len iterations in the update stream, we considered num_{upd} new articles).

Given this data set, we removed the stop words and applied stemming. We then selected the first n stories and identified the most frequent n keywords [1]. X_{ij} denotes occurrence of keyword j in story i. Intuitively, the low-rank decomposition of the data matrix X simultaneously cluster stories and keywords, resulting a co-clustering of the data matrix X. We moved the window at each iteration by inserting or deleting num_{upd} records of the story trace and recomputing the n most frequent keywords (meaning that num_{upd} many rows and columns are inserted and deleted). These correspond to row and column insertions/deletions on X.

5.2 Synthetic Data: Random Traces

We have also experimented with synthetic data sets where we could freely vary the characteristics of the data and updates to observe the accuracy and efficiency of our algorithms under different scenarios. For these experiments, we have created synthetic activity traces which we then converted into data matrices as before. Since the matrices for real data is sparse, we focus on dense matrices.

In particular, we have generated an initial n-length random sequence of 5 dimensional data, where each dimension has a value from 0 to 10. Given these n consecutive records in the trace, we have created a $n \times n$ initial matrix measuring pairwise Euclidean distances of the records in the sequence. Insertions in the random trace were generated by randomly picking numbers with exponential distribution, with the rate parameter, λ_{upd} (i.e., $prob(x) = exp\_dist(x, \lambda_{upd}) = \lambda_{upd} e^{-\lambda_{upd} x}$). Intuitively, if the rate parameter λ_{upd} is large, there is a higher likelihood of having more large amplitude changes. If the rate parameter λ_{upd} is low, there is a lower frequency of large amplitude changes in the trace.

As before, we enlarged or shrank X at each iteration by adding or deleting num_{upd} units of the random activity trace (meaning that num_{upd} many rows and columns are inserted to or deleted from into the matrix, X).

5.3 Evaluation Criteria and Competitors

We evaluate the LWI-SVD and LWI-SVD with Restart (LWI2-SVD) algorithms by comparing them to alternative approaches:

- *Full SVD and SVDS* – SVD is the full SV decomposition of the matrix, we used Matlab's `[U, S, V] = svd(X)` command for this. We also considered with Matlab's `[U, S, V] = svds(X,r)` command which returns the composition results for the top-r components, where r is the desired rank (SVDS tends to perform more efficiently than SVD when r is small and X is large and sparse);
- *Naive Incremental SVD* – this is our implementation of the Brand's algorithm described in [5], it involves a full SVD and pivoted QR based approximation is not leveraged (to implement LWI-SVD and LWI2-SVD, we use this implementation as the basis); and
- *SPIRIT* – this is the algorithm described in [13] which provides fast decompositions, but does not have various desirable properties of incremental SVD; including explicit data deletions and column insertions and deletions (for our experiments, we used the implementation obtained from [3]).

LWI-SVD family of the algorithms extend our implementation of the Brand's algorithm described in [5] along with the Algorithm 844 [4] obtained from [1].

As evaluation criteria, we use three metrics: reconstruction error overhead, execution time, and execution time gain:

- *average relative reconstruction error* (err_{rel}) – this accuracy measure is defined as

$$\frac{1}{len} \sum_{i=1}^{len} \frac{rec\_error(\hat{X}_{i,*}, X_i) - rec\_error(\hat{X}_{i,SVD}, X_i)}{rec\_error(\hat{X}_{i,SVD}, X_i)},$$

  where
  - len is the number of iterations (length of the stream),
  - $\hat{X}_{i,*}$ denotes the decomposition of the data matrix at time i obtained using the algorithm "$*$", and
  - $rec\_error(Y, X)$ denotes the reconstruction error of the decomposition Y against the data matrix X, measured in terms of $Frobenius$ norm.

  Note that a low-rank decomposition of X_i would lead to a reconstruction error, even if it is obtained using full SVD followed by selection of the top r components. Therefore, the denominator of the above term is not equal to 0.
- *absolute execution time* (t_{exec}) – this is the time, in seconds, that is required to complete len consecutive decompositions using the algorithm under consideration.
- *time gain* ($gain_{time}$) – the gain in time is the execution time measured against the execution time of the full SVD; i.e., $\frac{t_{exec,svd} - t_{exec,*}}{t_{exec,svd}}$.

All experiments were conducted using a 4-core Intel Core i5-2400, 3.10GHz, machine with 8GB memory, running 64-bit Windows 7 Enterprise. The codes were executed using Matlab 7.11.0(2010b).

5.4 Evaluation with the Default Settings

5.4.1 Real Trace Data Set

Figure 4 presents the accuracy and efficiency results for the real trace data for the default parameters reported in Table 1.

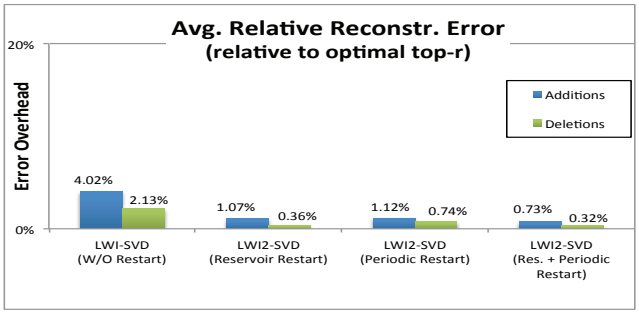
Accuracy. The first thing to note in Figure 4(a), which reports average relative reconstruction errors for the various versions of the LWI-SVD algorithm proposed in this paper, is that restarts discussed in Section 4 are highly effective in reducing the overall error. While both partial reconstruction-based and periodic restarts used in LWI2-SVD are effective in improving accuracy over the LWI-SVD (which does not use restarts), the best results are obtained when these are used together, bringing down the average relative reconstruction error to 0.3-0.7% of the low-rank decomposition obtained through full SVD.

The second thing to note in Figure 4(a) is that row/column insertions, which bring in new data into the matrix, results in larger relative reconstruction errors than row/column deletions. Note that, when both reservoir-based and periodic restarts are employed, the accuracy penalty relative to the low-rank decomposition of full SVD is negligibly low for both insertions and deletions.

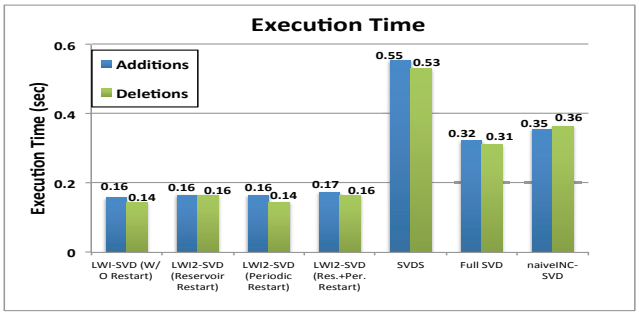
Efficiency. Figure 4(b) shows the efficiency results for this data set under the default parameter configuration.

The first thing to note is that there is minimal time difference between the LWI-SVD and LWI2-SVD algorithm. This indicates

[1] In these experiments, without loss of generality, we kept the matrix in square shape, i.e., $n = m$



(a) average relative reconstruction error



(b) execution time

Figure 4: Accuracy and efficiency for the *real trace data set* - *default settings*

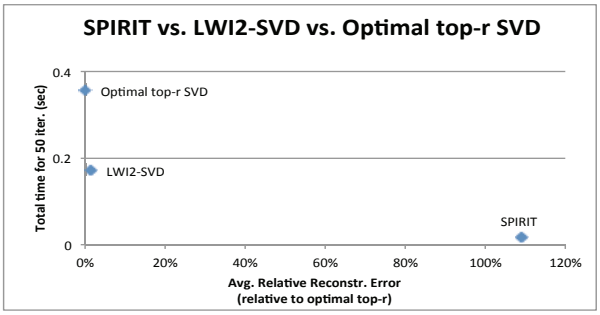


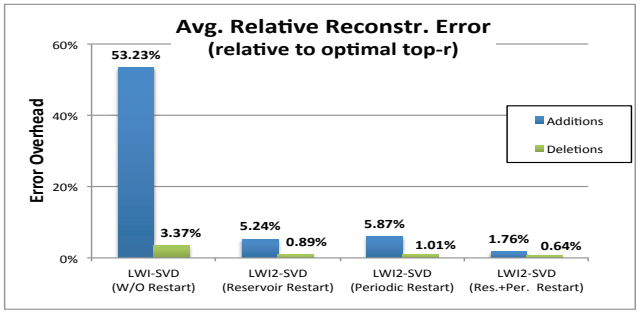
Figure 5: Accuracy and efficiency results for the *synthetic trace data set* for SVD, Spirit, and LWI2-SVD with periodic and on-demand refreshes

Table 2: Impact of setting $A = I$ in Section 3.1

	$A = I$	A is free	Impact
Rec. error	5.786	5.765	$+0.36\%$
Exec time	0.174 sec	0.197 sec	-11.71%

that the time overhead of reservoir maintenance and occasional on-demand full decompositions are negligible in the long run. Secondly, performing full SVD takes $\sim$ 75-100% more than the proposed LWI-SVD family of algorithms. Under this configuration, the naive incremental SVD takes a little more time than full SVD, as the basic algorithm reported in [5] involves a full SVD with same dimension as the original matrix and several matrix multiplications. Further-more, under this configuration, SVDS takes even longer than the full SVD.

Finally, a close look at the LWI-SVD family of algorithms indicates that insertions require slightly longer time to maintain than deletions. This is expected because, as discussed in Section 2.2.1, there is no need for computing R_A and R_B since they are all zero.



(a) average relative reconstruction error

(b) execution time

Figure 6: Accuracy and efficiency for *synthetic trace data set* - *default settings*

Impact of the QR-Elimination Optimization. In Section 3.1, we had discussed an optimization strategy whereby we eliminate one of the two expensive QR operations by forcing A to be equal to the identity matrix, I. As shown in Table 2, setting $A = I$ causes less than half percentage point impact on the accuracy; on the other hand, this optimization helps save close to 12% in execution time.

5.4.2 *Synthetic Trace Data Set*

Figure 6 presents results for the synthetic trace data set under the default parameter settings. The key observation from this figure is that the accuracy and efficiency results for the synthetic trace data set are very similar to the results for real trace data set, reported in Figure 4. The similarity is especially pronounced in the execution time results in Figure 6(b). This indicates that the execution time gains of the LWI-SVD family of algorithms (and to a certain degree, the accuracies they provide – especially with the help of periodic and reservoir-based restarts) are inherent properties of these algorithms rather than being highly data specific.

SPIRIT. Since the SPIRIT [13] algorithm approaches the problem differently (e.g. cannot directly handle deletions, cannot handle deletions/insertion of columns), we present it separately from the rest in Figure 5. For these experiments, we use a synthetic data trace that does not include any column insertions or deletions on the data matrix X. As the figure shows, SPIRIT algorithm works much faster than SVD or LWI2-SVD for the default configuration. However, this speed comes with a significant increase in the reconstruction error, relative to the optimal low-rank decomposition using SVD. In contrast, LWI2-SVD achieves an accuracy almost identical to the optimal, yet costs only half as much.

5.5 Impacts of Data and System Parameters

In this subsection, we evaluate the impacts of the various data and systems parameters on the efficiency and effectiveness of the LWI-SVD family of algorithms. As representative, we select the

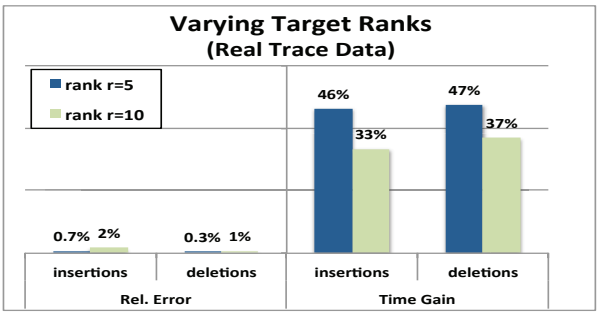


Figure 7: Accuracy and efficiency for *real trace data set* for different *rank*, r

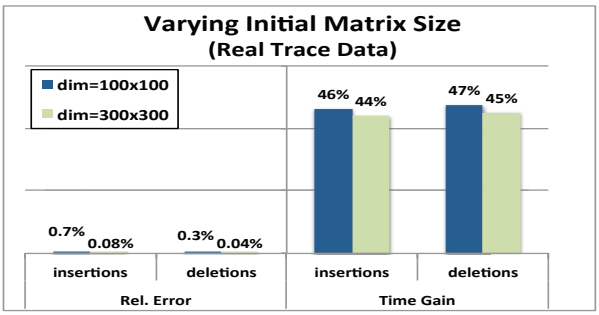


Figure 8: Accuracy and efficiency results for the *real trace data set* varying the *size*, dim, *of the initial (for insertions) / final (for deletions) matrix*

LWI2-SVD with the default parameters. We then vary, one-by-one, the various data and system parameters, and compare the results against the optimal SVD based rank-r decomposition. Since, as we have seen, the results are similar for real and synthetic data, for the most part we report the results with the real trace data. We use the synthetic trace only for experiments where we vary the strengths of the updates.

5.5.1 *Varying the Target Rank, r*

Figure 7 presents efficiency and accuracy results for the real trace data set where the target rank, r is varied. The results show that, as expected (due to the low-rank nature of the LWI-SVD family of algorithms), as the target rank increases, the time gain drops and the relative error rate slightly increases. The drop in time gains is because the incremental process involves a lot of matrix multiplications where the sizes of matrices are directly related to the target rank. This confirms the observation that LWI-SVD and LWI2-SVD are most effective when the target rank is low.

5.5.2 *Varying the Dimensions, dim, of the Matrix*

Figure 8 presents accuracy and efficiency results when we change the dimensions, dim, of the initial data matrix (for insertions) and the final data matrix (for deletions). Here, we see that increasing the size of matrix does not have a big impact on accuracy and efficiency.

5.5.3 *Varying the Rate of Updates, num_{upd}*

Figure 9 presents efficiency and accuracy results for the real trace data set where the number, num_{upd}, of row and column updates per each iteration is varied. The results indicate that, as expected, an increase in the number of updates per iteration impacts accuracy as well as efficiency. The slight impact on the accuracy is due to

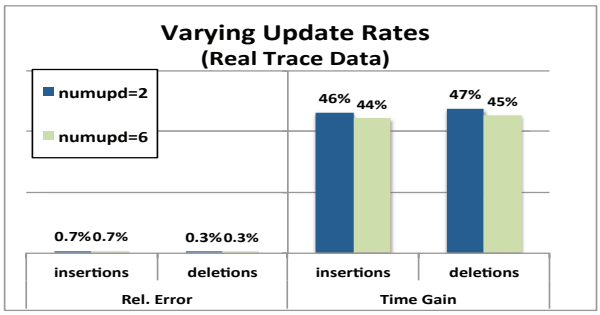


Figure 9: Accuracy and efficiency results for the *real trace data set* varying the *amount updates per iteration*, num_{upd}

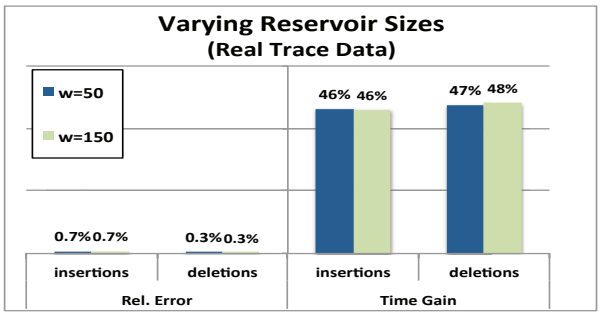


Figure 10: Accuracy and efficiency for *real trace data* varying *reservoir size*, w

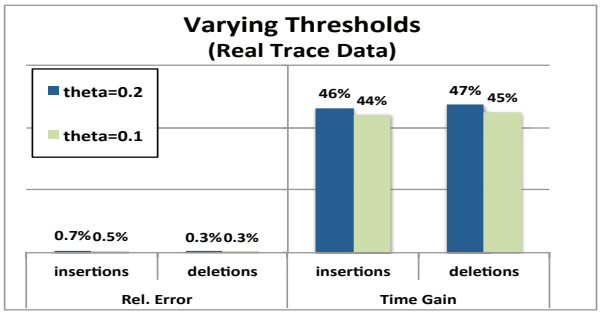


Figure 11: Accuracy and efficiency results for the *real trace data set* varying the *change threshold*, Θ, for on-demand restarts

the approximation nature of the algorithm. The impact on the time gain is due to more on-demand restarts.

5.5.4 *Varying the Reservoir Size, w*

Figure 10 presents efficiency and accuracy results for the real trace data set where the reservoir size, w, is varied. The results confirm that a larger reservoir (even only $\sim 1.5\%$ of the matrix) can help to trigger on-demand restarts more fairly, since larger reservoir has more accurate amortized error measuring.

5.5.5 *Varying the Change Threshold, Θ*

Figure 11 confirms that a slightly tighter threshold, $\Theta = 0.1$ instead of the default $\Theta = 0.2$ will trigger more on-demand restarts and thus can further reduce the error rates (which are already very low), with little impact on execution time gains.

5.5.6 *Varying the Restart Period, per*

Figure 12 confirms that increasing the number of restarts by reducing the restart period, per, may improve the final accuracy. However, unlike the on-demand restarts based on change detec-

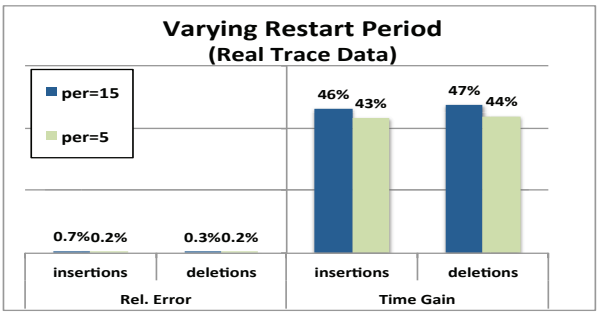


Figure 12: Accuracy and efficiency results for the *real trace data set* varying the *restart period,* per, for periodic restarts

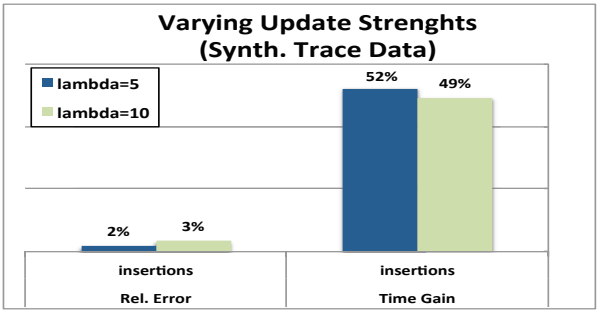


Figure 13: Accuracy and efficiency results for the *synthetic trace data set* varying the *update strength,* λ_{upd}

tion (shown in Figure 11), blindly increasing the frequency of the periodic restarts may negatively impact the time gain.

5.5.7 *Varying the Update Strength,* λ_{upd}

Finally, in Figure 13, we see the impact of the strength (in amplitude) of the incoming insertions. The figure shows that, when λ_{upd} increases, the LWI2-SVD algorithm adjusts its operation by scheduling more on-demand restarts at a cost of decreasing the time gain.

5.6 Scalability of LWI Algorithms

The results shown above are conducted with small window size, however, in some cases, we need large windows to monitor and analyze a large portion of the data. In this subsection, we analyze the scalability of LWI Algorithm by choosing large base number. Since we have shown that under the small base number condition, SVD out performs SVDS in execution time, however, when the base number is large, seeking a low rank deposition using SVDS is more efficient. Also, as we know that SVDS is very efficient when the data is sparse, but performs less efficient on dense data. We showed that LWI algorithm can concur this short coming when the data is dense. Recall in section 3.2.1, we showed that K is an arrow-like matrix which is very sparse, this leads to the efficiency by using pivoted QR compared to a direct SVDS on the dense data. Therefore, in the incremental maintenance of SVD on a dense matrix, we are actually seeking a second layer reduced rank approximation of a sparse matrix K. It is the main advantage of LWI algorithm compared to SVDS when the data is dense and the base dimension is large. Table 3 shows the execution time and error overhead results under a synthetic dense data, the results confirm that with big base number especially when the base is a thin and tall matrix, LWI algorithm can have advantages in execution time with negligible error overhead .

Table 3: Results for Large Dim

dim	LWI2 Exec. Time(s)	LWI2 Rel. Error	SVDS Exec. Time(s)
$1000 * 100$	8.2604	0.143%	15.91
$1000 * 1000$	6.8087	0.06%	10.839
$1500 * 1500$	17.483	0.03%	23.469
$2000 * 2000$	34.35	0.002%	41.491
$3000 * 3000$	96.577	0.00097%	93.622

6. CONCLUSIONS

In this paper, we presented a *Low-rank, Windowed, Incremental SVD* (LWI-SVD) algorithm, which relies on low-rank approximations to speed up incremental SVD updates. LWI-SVD algorithm also aggregates multiple row/column insertions and deletions to further reduce on-line processing cost. We also presented a *LWI-SVD with restarts* (LWI2-SVD) algorithm which performs periodic and change detection based on-demand refreshing of the decomposition to prevent accumulation of errors. Experiment results on real and synthetic data sets have shown that the LWI-SVD family of incremental SVD algorithms are highly efficient and accurate compared to alternative schemes under different settings.

7. REFERENCES

[1] Algorithm 844. Sparse Reduced-Rank Approximations to Sparse Matrices. *http://dl.acm.org/citation.cfm?id=1067972*. Downloaded 2012.

[2] Digg.com Data Set. *http://www.infochimps.com/datasets/diggcom-data-set*. Downloaded 2013.

[3] SPIRIT *http://www.cs.cmu.edu/afs/cs/project/spirit-1/www/*. Downloaded in 2012.

[4] M.W. Berry, S.A. Pulatova, and G.W. Stewart. Algorithm 844. Computing Sparse Reduced-Rank Approximations to Sparse Matrices. ACM Trans. Math. Softw. 31, 2, pp. 252-269, June 2005.

[5] M. Brand. Fast low-rank modifications of the thin singular value decomposition. Linear Algebra and its Appl., 415(1):20-30, 2006.

[6] S. Chandrasekaran, B.S. Manjunath, Y.F. Wang, J. Winkeler, and H. Zhang. An Eigenspace Update Algorithm for Image Analysis. Graphical Models and Image processing: GMIP, 59(5), 1997.

[7] S. Deerwester, S. Dumais, G.W. Furnas, T.K. Landauer, and R. Harshman. Indexing by Latent Semantic Analysis. Journal of the American Society for Information Science, 41(6):391-407, 1990.

[8] M. Gu and S. Eisenstat. Downdating the Singular Value Decomposition. SIAM J. Matrix Analysis and Applications, 1995.

[9] M. Gu and S.C. Eisenstat. A Stable and Fast Algorithm for Updating the Singular Value Decomposition. Tech. Report YALEU/DCS/RR-966, Department of Computer Science, Yale University, 1993.

[10] T. Kolda and B. Bader. Tensor Decompositions and Applications. SIAM Rev. 51, 3, 455-500. 2009.

[11] A. Levy and M. Lindenbaum. Sequential Karhunen-Loeve Basis Extraction and its Application to Images. IEEE Transactions on Image Processing, 9:1371-1374, 2000.

[12] G. O'Brien. Information Management Tools for Updating an SVD-Encoded Indexing Scheme, MS Thesis. 1994.

[13] S. Papadimitriou, J. Sun, and C. Faloutsos. Streaming Pattern Discovery in Multiple Time-Series. VLDB, pp. 697-708, 2005.

[14] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Incremental Singular Value Decomposition Algorithms for Highly Scalable Recommender Systems. ICIS, pp. 27–28, 2002.

[15] G.W. Steward. Four Algorithms for the Efficient Computation of Truncated Pivoted QR Approximations to a Sparse Matrix. Numerische Mathematik 83, 313-323, 1999.

[16] J. Sun, S. Papadimitriou, and C. Faloutsos. Online Latent Variable Detection in Sensor Networks, ICDE, 2005.

[17] D.I. Witter and M.W. Berry. Downdating the Latent Semantic Indexing Model for Conceptual Information Retrieval. The Computer Journal, 1998.

[18] J.S. Vitter. Random Sampling with a Reservoir. ACM Trans. Math. Softw. 11, 1, pp. 37-57, March 1985.

[19] H. Zha and H.D. Simon. On Updating Problems in Latent Semantic Indexing. SIAM J. Sci. Comput., 21(2):782-791, 1999.

[20] Z.A. Zhao and H. Liu. Spectral Feature Selection for Data Mining, Chapman and Hall/CRC Press, 2012.