Memory-efficient Centroid Decomposition for Long Time Series

Mourad Khayati
Department of Computer Science
University of Zurich
CH-8050 Zurich, Switzerland
mkhayati@ifi.uzh.ch

Michael Böhlen
Department of Computer Science
University of Zurich
CH-8050 Zurich, Switzerland
boehlen@ifi.uzh.ch

Johann Gamper
Faculty of Computer Science
Free University of Bozen-Bolzano
39100 Bozen-Bolzano, Italy
gamper@inf.unibz.it

***Abstract*—Real world applications that deal with time series data often rely on matrix decomposition techniques, such as the Singular Value Decomposition (SVD). The Centroid Decomposition (CD) approximates the Singular Value Decomposition, but does not scale to long time series because of the quadratic space complexity of the sign vector computation.**

In this paper, we propose a greedy algorithm, termed Scalable Sign Vector (SSV), to efficiently determine sign vectors for CD applications with long time series, i.e., where the number of rows (observations) is much larger than the number of columns (time series). The SSV algorithm starts with a sign vector consisting of only 1s and iteratively changes the sign of the element that maximizes the benefit. The space complexity of the SSV algorithm is linear in the length of the time series. We provide proofs for the scalability, the termination and the correctness of the SSV algorithm. Experiments with real world hydrological time series and data sets from the UCR repository validate the analytical results and show the scalability of SSV.

I. Introduction

The Centroid Decomposition (CD) has been introduced as an approximation of the Singular Value Decomposition (SVD). It decomposes an input matrix, $\mathbf{X}$, into the product of two matrices, $\mathbf{X} = \mathbf{L} \cdot \mathbf{R}^T$, where $\mathbf{L}$ is the loading matrix and $\mathbf{R}$ is the relevance matrix ($\mathbf{R}^T$ denotes the transpose of $\mathbf{R}$). Each loading and relevance vector is determined based on a maximal centroid value, $\max \|\mathbf{X}^T \cdot Z\|$, which is equal to the norm of the product between the transpose of the input matrix and the sign vector Z consisting of 1s and -1s. Finding the *maximizing sign vector* Z that maximizes the centroid value is therefore at the core of the CD method. The classical approach [1] enumerates all possible sign vectors and chooses the one that maximizes the centroid value. This approach has linear space complexity since no data structures other than the input matrix are needed. However, its runtime is exponential. A more efficient solution to determine the maximizing sign vector has been introduced by Chu and Funderlic [2] and has quadratic runtime. The drawback of this solution is a quadratic space complexity since a correlation/covariance matrix is needed in addition to the input matrix.

In this work, we address the scalability of the Centroid Decomposition technique for an $n \times m$ matrix, $\mathbf{X}$, that represents m time series with n observations each, where n is much larger than m (i.e., $n \gg m$). We propose a greedy algorithm, termed *Scalable Sign Vector (SSV)*, to compute the maximizing sign vector. The basic idea is as follows: instead of searching for the maximizing sign vector using all elements of $\mathbf{X}$, we search for it by rows of $\mathbf{X}$. First, a sign vector Z is initialized to contain only 1s as elements. Then, the algorithm iteratively updates the sign of the element in Z that increases $\|\mathbf{X}^T \cdot Z\|$ most. The relevant element can be determined efficiently by checking all elements of a weight vector V, which is derived from $\mathbf{X}$. Instead of enumerating all possible sign vectors, our strategy generates only the vectors that most increase $Z^T \cdot V$. At the end of this iterative process, the sign vector Z that yields the maximal centroid value is found. The SSV algorithm has quadratic time (worst case) and linear space complexity. Compared to the classical approach, SSV reduces the runtime of the CD method from exponential to quadratic while keeping the same linear space complexity. Compared to the most efficient algorithm [2], SSV keeps the same quadratic runtime complexity, but reduces the space complexity from quadratic to linear.

Matrix decomposition techniques are widely used for time series data in a variety of real world applications, such as data prediction, recommender systems, image compression, recovery of missing values, stocks, etc. In most of these applications, only very few and short time series can be considered for the analysis due to the computational complexity of current solutions for matrix decomposition. As a consequence, not all relevant information of the original set of time series is considered for the decomposition. This is an unfortunate limitation since it can be imperative to use long time series to improve data analysis [3]. For instance, in the recovery of missing values the most correlated time series are used to capture similar trends, and the use of longer time series improves the accuracy of the recovered values (as we will show in Section VI). Thus, scalable solutions that avoid an a priori segmentation of long time series are needed.

At the technical level, we provide an analysis and proofs of the correctness, the termination and the scalability of the SSV algorithm. The analytical results are confirmed by an in-depth empirical evaluation. In summary, the main contributions of this paper are the following:

- We propose a sign vector computation algorithm, called Scalable Sign Vector (SSV), that reduces the space complexity of the Centroid Decomposition technique from quadratic to linear, while keeping the same runtime complexity as the state-of-the-art solution.
- We prove that the space complexity of the SSV

algorithm increases linearly with the length of the time series.

- We prove that the computation performed by the SSV algorithm is strictly monotonic. We use the monotonicity property to prove the correctness of the proposed solution.
- We prove that the SSV algorithm terminates and performs at most n iterations.
- We present the results of an experimental evaluation of the efficiency and scalability of the SSV algorithm on real world hydrological data and on datasets from the UCR repository.

The remainder of the paper is organized as follows. Section II reviews related work. Preliminary concepts and definitions are provided in Section III. In Section IV, we present the SSV algorithm. Section V describes the main properties of the SSV algorithm. Section VI reports the results of our experiments. Section VII concludes the paper and discusses future work.

II. Related Work

The Centroid Decomposition (CD) has been introduced as an approximation of the Singular Value Decomposition [2]. It computes the centroid values, the loading vectors and the relevance vectors to approximate, respectively, the eigen values, the right singular vectors and the left singular vectors of SVD. Chu et al. [2], [4] prove that the CD approximation of SVD is the one that best minimizes the variance between corresponding elements. Thus, the variance between the centroid values computed by CD and the eigen values computed by SVD is minimal.

The most challenging part of the CD of a matrix $\mathbf{X}$ is the computation of the sign vector Z, consisting of 1s and -1s, that maximizes $\|\mathbf{X}^T \cdot Z\|$, where $\mathbf{X}^T$ is the transpose of $\mathbf{X}$ and $\|\cdot\|$ denotes the norm of a vector. The classical approach is based on the centroid method [5]. The centroid method uses a brute force search through an exponential number of sign vectors. Thus, the algorithm has exponential time and linear space complexity. This method has been used in various fields such as dimensionality reduction [6], peak shift detection [7], etc.

The most efficient algorithm to find the maximizing sign vector was introduced by Chu and Funderlic [2], which we refer to as *Quadratic Sign Vector* (QSV). It transforms the maximization problem from $\max \|\mathbf{X}^T \cdot Z\|$ to $\max (Z^T \cdot (\mathbf{X} \cdot \mathbf{X}^T) \cdot Z)$ and achieves a quadratic runtime complexity. The space complexity is quadratic as well due to the construction of $\mathbf{X} \cdot \mathbf{X}^T$. Fig. 1 illustrates the main data structures of the algorithm for an $n \times 3$ input matrix $\mathbf{X}$. Step 1 applies the transformation of the maximization problem, and Step 2 determines the maximizing sign vector. The set of all possible sign vectors can be considered as an n-dimensional hypercube, where each node represents a sign vector and is connected with all nodes representing a sign vector that differs in exactly one element. The QSV algorithm performs a traversal along the nodes of the hypercube, starting from the node that represents the sign vector $Z = [\text{-}1, \ldots, \text{-}1]^T$ until finding the node (and corresponding sign vector) that maximizes $Z^T \cdot (\mathbf{X} \cdot \mathbf{X}^T) \cdot Z$.

The Singular Value Decomposition (SVD) [8], [9], [10] is a widely used matrix decomposition technique. SVD performs the decomposition by finding the eigen values with their corresponding left and right singular vectors. SVD has been used for dimensionality reduction [11], [12], image compression [13], [14], missing values recovery [15], [16], etc. As most matrix decomposition techniques, SVD constructs a correlation/covariance matrix to find the eigen values and their corresponding singular vectors. Several algorithms have been proposed to make SVD for an $n \times m$ matrix faster for special cases of n and m. For instance, for $m > \frac{5}{3}n$, the runtime of SVD is reduced from $4n^2m + 8nm^2 + 9m^3$ to $2n^2m + 2m^3$ [17]. Less attention has been given to reduce the space complexity while keeping the same runtime complexity, which is the goal of this paper.

Rendle [18], [19] introduces Factorization Machines (FM) that perform a decomposition of large input matrices. Factorization Machines perform learning and prediction with input matrices of millions of values. For specific input matrices, the result obtained by the application of FM contains the decomposition matrices produced by SVD. The approach assumes the existence of repeating patterns in the input matrix. Each repeating pattern is represented by a block of data, and the decomposition is computed using these blocks of data.

Papalexakis et al. [20] present a scalable solution to compute tensor ($3d$ matrix in this work) decompositions [21]. More specifically, a parallel approach to compute the PARAFAC decomposition is proposed, which is a multidimensional generalization of SVD. The proposed solution works in three steps: create random samples of the input tensor, apply a parallel decomposition on each of them, and merge the result of each decomposition. This gives an approximation of the decomposition of the entire input tensor. The method scales linearly to millions of values. However, it is applicable only to sparse tensors, where more than 90% of the elements of the input tensor are equal to zero. Only the non-zero elements are used in the decomposition. In contrast, our solution performs the matrix decomposition for any type of input matrices.

Gemulla et al. [22] propose a large-scale matrix decomposition technique that, similar to CD, decomposes an input matrix into the product of two matrices. The proposed technique implements a scalable distributed Stochastic Gradient Descent (SGD) method [23]. The latter computes a loss function that minimizes an error value between the input matrix and the product of the two matrices. The method works as follows. First, the input matrix is partitioned into blocks that are distributed across a MapReduce cluster. Then, the loss function is computed as the sum of local loss functions, each of which is computed in parallel over a single data block. This technique exploits the fact that the computation of local loss functions can be swapped without changing the final result of the decomposition. This is possible since each local loss function is computed over a different row and/or column of the input matrix. This technique scales to large input matrices consisting of millions of rows and columns. The technique we propose in this paper cannot be distributed since it computes a sequence of vectors, where each vector relies on the previous one.

Li et al. [24] follow the same idea to distribute the computation of a matrix decomposition across a MapReduce

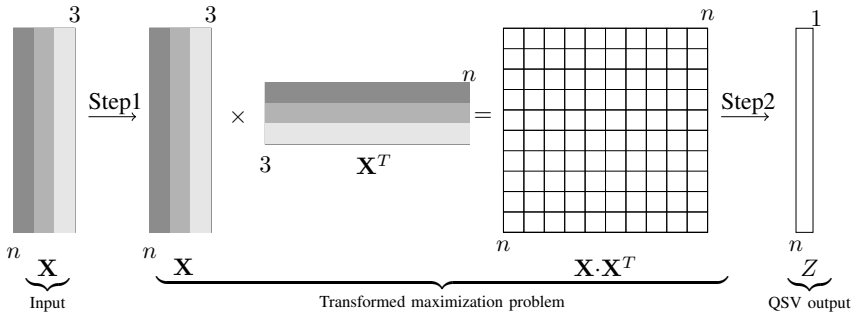


Fig. 1. Illustration of the main data structures used by the QSV algorithm.

cluster. They propose a parallelizable computation of the SGD method using Resilient Distributed Datasets (RDDs) [25], but investigate the case when the individual blocks do not fit into the main memory of a node. The proposed solution is based on a hash table that stores partitions of data blocks in memory. It scales to large data sets containing millions of values. The work uses the fact that the decomposition can be performed in parallel over separate rows and/or columns of data. Our solution cannot be computed over separate rows and/or columns of the input matrix since the sequence of computed vectors requires the use of the entire matrix.

The methods described in Gemulla et al. [22] and Li et al. [24] implement a scalable SGD method. They do not assume any constraints about the input matrix, but compute a decomposition that is different from the one produced by SVD. The SGD method minimizes a given error value, which is different from CD. This approach makes SGD suitable for applications where an error value needs to be minimized, such as in recommender systems [26] used in Netflix [27], [28], [29] or MovieLens [30].

The solution proposed in this paper describes a scalable implementation of the CD matrix decomposition technique that approximates SVD. Our solution uses the entire input matrix for the computation and can therefore not be computed over separate blocks of rows and/or columns in parallel. Instead of distributing the decomposition across clusters, we propose to reduce the space complexity of the decomposition method.

III. Preliminaries

A. Notations

Bold upper-case letters refer to matrices, regular font upper-case letters to vectors (rows and columns of matrices) and lower-case letters to elements of vectors/matrices. For example, $\mathbf{X}$ is a matrix, X_{i*} is the i-th row of $\mathbf{X}$, X_{*i} is the i-th column of $\mathbf{X}$, $(X_{i*})^T$ is the transpose of X_{i*} and x_{ij} is the j-th element of X_{i*}.

A *time series* $X = \{(t_1, v_1), \ldots, (t_n, v_n)\}$ is a set of n temporal values v_i that are ordered with respect to the timestamps t_i. We assume time series with aligned timestamps (possibly after a preprocessing step). Thus, in the rest of the paper we omit the timestamps and write a time series $X_1 = \{(0, 2), (1, 0), (2, \text{-}4)\}$ as $X_1 = \{2, 0, \text{-}4\}$.

An $n \times m$ matrix $\mathbf{X} = [X_{*1}| \ldots |X_{*m}]$ contains as columns m time series X_{*j} and as rows n values for each time series. Our database contains up to 200 real world hydrological time series with each of them containing up to 120k values. Thus, we consider matrices where the length of time series n is much larger than the number of time series m, i.e., $n \gg m$.

A *sign vector* $Z \in \{1, \text{-}1\}^n$ is a sequence $[z_1, \ldots, z_n]$ of n elements, i.e., $|z_i| = 1$ for $i = 1, \ldots, n$.

B. Centroid Decomposition

The *Centroid Decomposition (CD)* is a decomposition technique that decomposes an $n \times m$ matrix, $\mathbf{X} = [X_{*1}| \ldots |X_{*m}]$, into an $n \times m$ loading matrix, $\mathbf{L} = [L_{*1}| \ldots |L_{*m}]$, and an $m \times m$ relevance matrix, $\mathbf{R} = [R_{*1}| \ldots |R_{*m}]$, i.e.,

$$\mathbf{X} = \mathbf{L} \cdot \mathbf{R}^T = \sum_{i=1}^{m} L_{*i} \cdot (R_{*i})^T,$$

where $\mathbf{R}^T$ denotes the transpose of $\mathbf{R}$.

Algorithm 1: CD($\mathbf{X}$, n, m)

Input: $n \times m$ matrix $\mathbf{X}$
Output: $\mathbf{L}$, $\mathbf{R}$

$L = R = []$;
for $i = 1$ **to** m **do**
  $Z = FindSignVector(\mathbf{X}, n, m)$;
  $C_{*i} = \mathbf{X}^T \cdot Z$;
  $R_{*i} = \frac{C_{*i}}{\|C_{*i}\|}$;
  $\mathbf{R} = Append(\mathbf{R}, R_{*i})$;
  $L_{*i} = \mathbf{X} \cdot R_{*i}$;
  $\mathbf{L} = Append(\mathbf{L}, L_{*i})$;
  $\mathbf{X} := \mathbf{X} - L_{*i} \cdot R_{*i}^T$;
return $\mathbf{L}$, $\mathbf{R}$

Algorithm 1 computes the CD of an input matrix $\mathbf{X}$ into matrices $\mathbf{L}$ and $\mathbf{R}$. At each iteration i, function FindSignVector() determines the sign vector Z that yields the maximal centroid value $\|\mathbf{X}^T \cdot Z\|$ (where $\|\mathbf{X}^T \cdot Z\|$ is equal to the square root of the squared elements of $\mathbf{X}^T \cdot Z$). We call Z the *maximizing sign vector*. Next, the centroid vector, C_{*i}, and the centroid value, $\|C_{*i}\|$, are computed. Finally, the vectors L_{*i} and R_{*i} are computed and added as columns to, respectively, $\mathbf{L}$ and $\mathbf{R}$. In order to eliminate duplicate vectors, the next loading vectors, L_{*i+1}, and relevance vectors, R_{*i+1}, are computed from $\mathbf{X} - L_{*i} \cdot R_{*i}^T$. The algorithm terminates when m centroid values and m loading and relevance vectors are found.

Example 1: Consider a matrix $\mathbf{X}$ and two sign vectors:

$$\mathbf{X}=\begin{bmatrix}2 & -2\\ 0 & 3\\ -4 & 2\end{bmatrix} \quad \mathbf{X}^T=\begin{bmatrix}2 & 0 & -4\\ -2 & 3 & 2\end{bmatrix} \quad Z_1=\begin{bmatrix}1\\ -1\\ 1\end{bmatrix} \quad Z_2=\begin{bmatrix}-1\\ 1\\ 1\end{bmatrix}$$

The centroid values for the two sign vectors are computed as $\|\mathbf{X}^T{\cdot}Z_1\| = \sqrt{(-2)^2+(-3)^2} = \sqrt{13}$ and $\|\mathbf{X}^T{\cdot}Z_2\| = \sqrt{(-6)^2+7^2} = \sqrt{85}$. Since $\|\mathbf{X}^T{\cdot}Z_2\| > \|\mathbf{X}^T{\cdot}Z_1\|$, Z_2 is the maximizing sign vector (among the two sign vectors).

C. Application of CD

The following example illustrates how to interpret the Centroid Decomposition of a matrix of m time series. Let $F = \{f_1, \ldots, f_m\}$ be the set of m factors that (most) influence the values in the time series.

Example 2: Consider a 2×3 matrix $\mathbf{X} = \{X_1, X_2\}$ that consists of two time series $X_1 = \{2, 0, -4\}$ and $X_2 = \{-2, 3, 2\}$. X_1 is the temperature in Zurich, and X_2 is the temperature in Basel. The CD method decomposes $\mathbf{X}$ by finding the loading and the relevance vectors with respect to each time series as shown in Fig. 2.

$$\mathbf{X} = \begin{bmatrix} 2 & -2\\ 0 & 3\\ -4 & 2\end{bmatrix}$$

$$\text{CD}(\mathbf{X}) = \underbrace{\begin{bmatrix} -2.820 & 0.217\\ 2.278 & 1.952\\ 4.122 & -1.735\end{bmatrix}}_{\mathbf{L}}, \underbrace{\begin{bmatrix} -0.651 & 0.759\\ 0.759 & 0.651\end{bmatrix}}_{\mathbf{R}}$$

such that

$$\mathbf{X} = \begin{bmatrix} 2 & -2\\ 0 & 3\\ -4 & 2\end{bmatrix} = \underbrace{\begin{bmatrix} -2.820 & 0.217\\ 2.278 & 1.952\\ 4.122 & -1.735\end{bmatrix}}_{\mathbf{L}} \times \underbrace{\begin{bmatrix} -0.651 & 0.759\\ 0.759 & 0.651\end{bmatrix}}_{\mathbf{R}^T}$$

Fig. 2. Example of Centroid Decomposition.

If $F = \{nbrSunnyHours, amntRain\}$ and if each temperature time series is mainly influenced by the two factors of F, the Centroid Decomposition shows how to obtain the temperature values in a specific city using these two factors. For instance, the first value of the temperature in Zurich shown in gray color (i.e., 2) is obtained using a loading value of -2.820 for $nbrSunnyHours$ with a relevance value of -0.651 to which we add a loading value of 0.217 for $amntRain$ with a relevance value of 0.759.

The result of the decomposition in Example 2 can be used to recover missing values. Let's assume the second value of the temperature in Basel is missing. The first step of the recovery process is to initialize the missing value using a classical imputation technique, e.g., linear interpolation. Then, we apply the Centroid Decomposition to learn the loading and relevance values of the two factors to refine the initialized value. The refined values better approximate the missing value. We show the result of the recovery based on Centroid Decomposition in Section VI-B5.

IV. Scalable Sign Vector

This section presents a scalable sign vector (SSV) computation technique, which has the same quadratic runtime complexity as the QSV algorithm [2], but requires only linear space. The core of the solution is the transformation of the maximization of the centroid values $\|\mathbf{X}^T{\cdot}Z\|$ into a new and equivalent maximization problem that can be computed efficiently.

A. Overview and Data Structures

Fig. 3 illustrates the SSV computation method. We transform the maximization of the centroid values to a new maximization problem that is based on a sign vector Z and a weight vector V that is derived from $\mathbf{X}$. More specifically, a sequence of vector pairs $V^{(k)}$ and $Z^{(k)}$ is iteratively computed, beginning with $Z^{(1)} = [1, \ldots, 1]^T$. In each iteration, the sign vector is changed at the position that maximizes the product of the two vectors. The last sign vector $Z^{(k)}$ $(1 \leq k \leq n)$ is the maximizing sign vector.

$$\underbrace{\begin{bmatrix} 2 & -2\\ 0 & 3\\ -4 & 2\end{bmatrix}}_{\mathbf{X}} \rightarrow \underbrace{\begin{bmatrix} 1\\ 1\\ 1\end{bmatrix}}_{Z^{(1)}} \underbrace{\begin{bmatrix} -18\\ 0\\ -6\end{bmatrix}}_{V^{(1)}} \underbrace{\begin{bmatrix} -1\\ 1\\ 1\end{bmatrix}}_{Z^{(2)}} \underbrace{\begin{bmatrix} -18\\ 12\\ 18\end{bmatrix}}_{V^{(2)}} \rightarrow \underbrace{\begin{bmatrix} -1\\ 1\\ 1\end{bmatrix}}_{Z^{(2)}}$$

Input — SSV computation — SSV output

Fig. 3. Illustration of SSV.

Fig. 4 illustrates the Centroid Decomposition of an input matrix $\mathbf{X}$ using SSV. In the first iteration of Algorithm 1, we use $\mathbf{X}$ and SSV to derive the first maximizing sign vector. We then compute vector L_{*1} and R_{*1} according to Algorithm 1. In the second iteration, we update matrix $\mathbf{X}$ to $\mathbf{X}' = \mathbf{X} - L_{*1}{\cdot}R_{*1}^T$ and repeat the process, i.e., derive the second maximizing sign vector Z' and compute vectors L_{*2} and R_{*2}.

$$\underbrace{\begin{bmatrix} 2 & -2\\ 0 & 3\\ -4 & 2\end{bmatrix}}_{\mathbf{X}} \rightarrow \underbrace{\begin{bmatrix} -1\\ 1\\ 1\end{bmatrix}}_{Z} \rightarrow \underbrace{\begin{bmatrix} -2.820\\ 2.278\\ 4.122\end{bmatrix}}_{L_{*1}} \underbrace{\begin{bmatrix} -0.651\\ 0.759\end{bmatrix}}_{R_{*1}} \rightarrow$$

(a) Iteration 1

$$\rightarrow \underbrace{\begin{bmatrix} 0.164 & 0.141\\ 1.482 & 1.270\\ -1.317 & -1.129\end{bmatrix}}_{\mathbf{X}'} \rightarrow \underbrace{\begin{bmatrix} 1\\ 1\\ -1\end{bmatrix}}_{Z'} \rightarrow \underbrace{\begin{bmatrix} 0.217\\ 1.952\\ -1.735\end{bmatrix}}_{L_{*2}} \underbrace{\begin{bmatrix} 0.759\\ 0.651\end{bmatrix}}_{R_{*2}}$$

(b) Iteration 2

Fig. 4. Illustration of Centroid Decomposition using SSV.

B. Transformation of the Maximization Problem

In this section, we present a transformation of the maximization of $\|\mathbf{X}^T{\cdot}Z\|$ into a new and equivalent maximization problem and we show that the new maximization problem can be efficiently computed with linear space complexity.

The following auxiliary function is used: $diag^{=0}(\mathbf{X})$ sets the diagonal values of an $n \times n$ matrix $\mathbf{X}$ to 0.

The following lemma introduces a maximization equivalence, which states that maximizing $\|\mathbf{X}^T{\cdot}Z\|$ over all possible

sign vectors Z is equivalent to maximizing the product of Z^T and the vector $V = diag^{\neq 0}(\mathbf{X}\cdot\mathbf{X}^T)\cdot Z$. This maximization equivalence will be used to define the new maximization problem.

Lemma 1 (Maximization Equivalence): Let matrix $\mathbf{X} = [X_{*1}|\dots|X_{*m}]$ be an $n \times m$ matrix and V be the vector $V = diag^{\neq 0}(\mathbf{X}\cdot\mathbf{X}^T)\cdot Z$. The following equivalence holds:

$$\underset{Z\in\{1,-1\}^n}{\arg\max}\ \|\mathbf{X}^T\cdot Z\| \equiv \underset{Z\in\{1,-1\}^n}{\arg\max}\ Z^T\cdot V.$$

Proof: We expand both sides of the equivalence and show that the expanded expressions are equivalent.

The transformation of the expression on the left-hand side yields

$$\begin{aligned}
&\underset{Z\in\{-1,1\}^n}{\arg\max}\ \|\mathbf{X}^T\cdot Z\| \equiv \\
&\quad\equiv \underset{Z\in\{-1,1\}^n}{\arg\max}\ \|\mathbf{X}^T\cdot Z\|^2 \\
&\quad\equiv \underset{Z\in\{-1,1\}^n}{\arg\max}\ ((\sum_{i=1}^{n} x_{i1}\times z_i)^2 + \dots + (\sum_{i=1}^{n} x_{im}\times z_i)^2) \\
&\quad\equiv \underset{Z\in\{-1,1\}^n}{\arg\max}\ ((\sum_{i=1}^{n} \tilde{x}_{i1})^2 + \dots + (\sum_{i=1}^{n} \tilde{x}_{im})^2),
\end{aligned}$$

where $\tilde{x}_{ij} = x_{ij} \times z_i$ for $j = 1,\dots,m$. Notice that the first step takes the square of the norm, which has no impact on the vector Z that maximizes the norm. Next, we use the square of sum rule

$$(\sum_{i=1}^{n} x_i)^2 = \sum_{i=1}^{n} x_i^2 + 2\times\sum_{j=2}^{n}\sum_{i=1}^{j-1} x_i\times x_j$$

and transform the above expression into

$$\begin{aligned}
&\underset{Z\in\{-1,1\}^n}{\arg\max}\ \|\mathbf{X}^T\cdot Z\| \equiv \\
&\quad\equiv \underset{Z\in\{-1,1\}^n}{\arg\max}\ (\sum_{i=1}^{n} \tilde{x}_{i1}^2 + 2\times\sum_{j=2}^{n}\sum_{i=1}^{j-1} \tilde{x}_{i1}\times\tilde{x}_{j1} + \\
&\qquad\qquad \vdots \\
&\qquad\qquad \sum_{i=1}^{n} \tilde{x}_{im}^2 + 2\times\sum_{j=2}^{n}\sum_{i=1}^{j-1} \tilde{x}_{im}\times\tilde{x}_{jm}).
\end{aligned}$$

Since $\tilde{x}_{ij} = x_{ij} \times z_i$ with $z_i \in \{-1, 1\}$, we have $\tilde{x}_{ij}^2 = x_{ij}^2$. That is, the terms $\sum_{i=1}^{n} \tilde{x}_{ij}^2$ for $j = 1,\dots,m$ in the above expression are constant and independent of the sign vector Z. Therefore, they can be removed from the maximization problem, which yields

$$\begin{aligned}
&\underset{Z\in\{-1,1\}^n}{\arg\max}\ \|\mathbf{X}^T\cdot Z\| \equiv \\
&\quad\equiv \underset{Z\in\{-1,1\}^n}{\arg\max}\ (2\times(\underbrace{\sum_{j=2}^{n}\sum_{i=1}^{j-1} \tilde{x}_{i1}\times\tilde{x}_{j1} + \dots + \sum_{j=2}^{n}\sum_{i=1}^{j-1} \tilde{x}_{im}\times\tilde{x}_{jm}}_{\sum_{k=1}^{m}\sum_{j=2}^{n}\sum_{i=1}^{j-1}\tilde{x}_{ik}\times\tilde{x}_{jk}})) \\
&\quad\equiv \underset{Z\in\{-1,1\}^n}{\arg\max}\ (2\times\sum_{k=1}^{m}\sum_{j=2}^{n}\sum_{i=1}^{j-1} \tilde{x}_{ik}\times\tilde{x}_{jk}). \qquad (1)
\end{aligned}$$

Next, we transform the expression on the right-hand side. From the definition of V we get

$$\underset{Z\in\{-1,1\}^n}{\arg\max}\ Z^T\cdot V \equiv \underset{Z\in\{-1,1\}^n}{\arg\max}\ (Z^T\cdot diag^{\neq 0}(\mathbf{X}\cdot\mathbf{X}^T)\cdot Z).$$

The matrix representation of $diag^{\neq 0}(\mathbf{X}\cdot\mathbf{X}^T)$ using the rows of $\mathbf{X}$ is given as

$$diag^{\neq 0}(\mathbf{X}\cdot\mathbf{X}^T) = \begin{bmatrix} 0 & X_{1*}\cdot(X_{2*})^T & \cdots & X_{1*}\cdot(X_{n*})^T \\ X_{2*}\cdot(X_{1*})^T & 0 & \cdots & X_{2*}\cdot(X_{n*})^T \\ \vdots & \vdots & \ddots & \vdots \\ X_{n*}\cdot(X_{1*})^T & X_{n*}\cdot(X_{2*})^T & \cdots & 0 \end{bmatrix}.$$

We use this representation and transform the argument of the $\arg\max$ function as follows:

$$\begin{aligned}
&Z^T\cdot diag^{\neq 0}(\mathbf{X}\cdot\mathbf{X}^T)\cdot Z = \\
&= Z^T\cdot\begin{bmatrix} 0 & X_{1*}\cdot(X_{2*})^T & \cdots & X_{1*}\cdot(X_{n*})^T \\ X_{2*}\cdot(X_{1*})^T & 0 & \cdots & X_{2*}\cdot(X_{n*})^T \\ \vdots & \vdots & \ddots & \vdots \\ X_{n*}\cdot(X_{1*})^T & X_{n*}\cdot(X_{2*})^T & \cdots & 0 \end{bmatrix}\cdot Z \\
&= Z^T\cdot\begin{bmatrix} 0 + z_2\times(X_{1*}\cdot(X_{2*})^T) + \dots + z_n\times(X_{1*}\cdot(X_{n*})^T) \\ z_1\times(X_{2*}\cdot(X_{1*})^T) + 0 + \dots + z_n\times(X_{2*}\cdot(X_{n*})^T) \\ \vdots \\ z_1\times(X_{n*}\cdot(X_{1*})^T) + z_2\times(X_{n*}\cdot(X_{2*})^T) + \dots + 0 \end{bmatrix} \\
&= z_1\times(0 + z_2\times(X_{1*}\cdot(X_{2*})^T) + \dots + z_n\times(X_{1*}\cdot(X_{n*})^T)) + \\
&\quad z_2\times(z_1\times(X_{2*}\cdot(X_{1*})^T) + 0 + \dots + z_n\times(X_{2*}\cdot(X_{n*})^T)) + \\
&\qquad\qquad \vdots \\
&\quad z_n\times(z_1\times(X_{n*}\cdot(X_{1*})^T) + z_2\times(X_{n*}\cdot(X_{2*})^T) + \dots + 0).
\end{aligned}$$

The vector products in the above expression are replaced by a sum, i.e., $X_{i*}\cdot(X_{j*})^T = \sum_{k=1}^{m} x_{ik}\times x_{jk}$, which gives

$$\begin{aligned}
&Z^T\cdot diag^{\neq 0}(\mathbf{X}\cdot\mathbf{X}^T)\cdot Z = \\
&= z_1\times(0 + z_2\times\sum_{k=1}^{m} x_{1k}\times x_{2k} + \dots + z_n\times\sum_{k=1}^{m} x_{1k}\times x_{nk}) + \\
&\quad z_2\times(z_1\times\sum_{k=1}^{m} x_{2k}\times x_{1k} + 0 + \dots + z_n\times\sum_{k=1}^{m} x_{2k}\times x_{nk}) + \\
&\qquad \vdots \\
&\quad z_n\times(z_1\times\sum_{k=1}^{m} x_{nk}\times x_{1k} + z_2\times\sum_{k=1}^{m} x_{nk}\times x_{2k} + \dots + 0).
\end{aligned}$$

Finally, we push the elements of the sign vector into the sum and replace $z_i \times x_{ik}$ by $\tilde{x}_{ik}$, which gives

$$
\begin{aligned}
&Z^T{\cdot}diag^{\neq 0}(\mathbf{X}{\cdot}\mathbf{X}^T){\cdot}Z = \\
&= 0 + \sum_{k=1}^{m} \tilde{x}_{1k}{\times}\tilde{x}_{2k} + \cdots + \sum_{k=1}^{m} \tilde{x}_{1k}{\times}\tilde{x}_{nk} + \\
&\quad \sum_{k=1}^{m} \tilde{x}_{2k}{\times}\tilde{x}_{1k} + 0 + \cdots + \sum_{k=1}^{m} \tilde{x}_{2k}{\times}\tilde{x}_{nk} + \\
&\quad \vdots \\
&\quad \sum_{k=1}^{m} \tilde{x}_{nk}{\times}\tilde{x}_{1k} + \sum_{k=1}^{m} \tilde{x}_{nk}{\times}\tilde{x}_{2k} + \cdots + 0 \\
&= 2 \times \sum_{k=1}^{m} \sum_{j=2}^{n} \sum_{i=1}^{j-1} \tilde{x}_{ik}{\times}\tilde{x}_{jk}.
\end{aligned}
$$

We insert this expression in the $\arg\max$ function, which gives Equation (1). ■

Lemma 1 forms the basis for a new and equivalent maximization problem. Instead of maximizing $\|\mathbf{X}^T{\cdot}Z\|$, the product $Z^T{\cdot}V$ is maximized over all sign vectors $Z \in \{1,\text{-}1\}^n$. Since the computation of $V = diag^{\neq 0}(\mathbf{X}{\cdot}\mathbf{X}^T){\cdot}Z$ has quadratic space complexity (due to the construction of the matrix $\mathbf{X}{\cdot}\mathbf{X}^T$), we proceed by showing how to avoid this product and how to compute V directly from $\mathbf{X}$.

Lemma 2: Let $\mathbf{X} = [X_{*1}|\ldots|X_{*m}]$ be an $n \times m$ matrix and v_i be the i-th element of the vector $V = diag^{\neq 0}(\mathbf{X}{\cdot}\mathbf{X}^T){\cdot}Z$. Then, the following holds:

$$v_i = z_i \times (z_i \times X_{i*}{\cdot}Z^T{\cdot}\mathbf{X} - X_{i*}{\cdot}(X_{i*})^T).$$

Proof: We use the matrix representation of $diag^{\neq 0}(\mathbf{X}{\cdot}\mathbf{X}^T)$ to compute V as follows:

$$
\begin{aligned}
&diag^{\neq 0}(\mathbf{X}{\cdot}\mathbf{X}^T){\cdot}Z = \\
&= \begin{bmatrix} 0 & X_{1*}{\cdot}(X_{2*})^T & \cdots & X_{1*}{\cdot}(X_{n*})^T \\ X_{2*}{\cdot}(X_{1*})^T & 0 & \cdots & X_{2*}{\cdot}(X_{n*})^T \\ \vdots & \vdots & \ddots & \vdots \\ X_{n*}{\cdot}(X_{1*})^T & X_{n*}{\cdot}(X_{2*})^T & \cdots & 0 \end{bmatrix} {\cdot} Z \\
&= \begin{bmatrix} X_{1*}{\cdot}(0 + z_2{\times}(X_{2*})^T + \cdots + z_n{\times}(X_{n*})^T) \\ X_{2*}{\cdot}(z_1{\times}(X_{1*})^T + 0 + \cdots + z_n{\times}(X_{n*})^T) \\ \vdots \\ X_{n*}{\cdot}(z_1{\times}(X_{1*})^T + z_2{\times}(X_{2*})^T + \cdots + 0) \end{bmatrix} \\
&= \begin{bmatrix} z_1 \times (z_1 \times X_{1*}{\cdot}\sum_{j=1}^{n}(z_j \times (X_{j*})^T) - X_{1*}{\cdot}(X_{1*})^T) \\ z_2 \times (z_2 \times X_{2*}{\cdot}\sum_{j=1}^{n}(z_j \times (X_{j*})^T) - X_{2*}{\cdot}(X_{2*})^T) \\ \vdots \\ z_n \times (z_n \times X_{n*}{\cdot}\sum_{j=1}^{n}(z_j \times (X_{j*})^T) - X_{n*}{\cdot}(X_{n*})^T) \end{bmatrix}
\end{aligned}
$$

Since we have that $Z^T{\cdot}\mathbf{X} = \sum_{j=1}^{n}(z_j{\times}(X_{j*})^T)$, it follows that $v_i = z_i \times (z_i \times X_{i*}{\cdot}Z^T{\cdot}\mathbf{X} - X_{i*}{\cdot}(X_{i*})^T)$. ■

Based on Lemma 2 we show that the space complexity of computing vector V (i.e., $diag^{\neq 0}(\mathbf{X}{\cdot}\mathbf{X}^T){\cdot}Z$) is linear in the number of rows of $\mathbf{X}$.

Lemma 3 (Linear Space): For an $n \times m$ matrix $\mathbf{X}$, the computation of $V = diag^{\neq 0}(\mathbf{X}{\cdot}\mathbf{X}^T){\cdot}Z$ has $O(n)$ space complexity.

Proof: The result of $\sum_{j=1}^{n}(z_j \times (X_{j*})^T)$ is computed by keeping in memory a single row X_{j*} and one element of Z at a time, which requires $O(m)$ space. This sum is computed only once. To compute the individual elements $v_i = z_i \times (X_{i*}{\cdot}Z^T{\cdot}\mathbf{X} - X_{i*}{\cdot}(X_{i*})^T)$ of the result vector of $diag^{\neq 0}(\mathbf{X}{\cdot}\mathbf{X}^T){\cdot}Z$, $\mathbf{X}$ is read again, one row at a time. The result vector has length n. Since $n \gg m$, the space complexity of $diag^{\neq 0}(\mathbf{X}{\cdot}\mathbf{X}^T){\cdot}Z$ is $O(n)$. ■

C. Computation of Maximizing Sign Vectors

We present now an algorithm with linear space complexity to compute the maximizing sign vector Z according to the maximization problem introduced in Lemma 1. The basic idea is as follows. We begin with the sign vector $Z = [1,\ldots,1]^T$ and iteratively change the sign of one element in Z that increases $Z^T{\cdot}V$ most. The algorithm terminates when $Z^T{\cdot}V$ cannot be increased further with this strategy.

Algorithm 2: SSV($\mathbf{X}$, n, m)

Input: $n \times m$ matrix $\mathbf{X}$
Output: maximizing sign vector $Z^T = [z_1,\ldots,z_n]$

```
pos = 0;
repeat
    // Change sign
    if pos = 0 then Z^T = [1,...,1];
    else change the sign of z_pos;
    // Determine S and V
    S = Σ_{i=1}^{n} (z_i × (X_i*)^T);
    V = [];
    for i = 1 to n do
        v_i = z_i × (z_i × X_i* · S − X_i* · (X_i*)^T);
        Insert v_i in V;
    // Search next element
    val = 0, pos = 0;
    for i = 1 to n do
        if (z_i × v_i < 0) then
            if |v_i| > val then
                val = v_i;
                pos = i;
until pos = 0;
return Z;
```

Algorithm 2 implements this strategy and computes the maximizing sign vector. Note that V is computed directly from $\mathbf{X}$ by reading the matrix row by row, one row at a time: first to compute the intermediate vector S and then to compute the individual elements of V. We search for the index (pos) of the element $v_i \in V$ with the largest absolute value such that v_i and $z_i \in Z$ have different signs, i.e., $z_i \times v_i < 0$. If such an element is found, the sign of z_i is changed. A new vector V is computed, which is different from the vector in the previous iteration due to the sign change. The iteration

terminates when the signs of all corresponding elements in V and Z are the same. The vector Z in the final iteration is the maximizing sign vector that maximizes $Z^T \cdot V$.

Example 3: To illustrate the computation of the sign vector using Algorithm 2, consider the input matrix of our running example, i.e.,

$$\mathbf{X} = \begin{bmatrix} 2 & -2 \\ 0 & 3 \\ -4 & 2 \end{bmatrix}.$$

First, Z is initialized with 1s, and S and V are computed:

$$S = \begin{bmatrix} 2 \\ -2 \end{bmatrix} + \begin{bmatrix} 0 \\ 3 \end{bmatrix} + \begin{bmatrix} -4 \\ 2 \end{bmatrix} = \begin{bmatrix} -2 \\ 3 \end{bmatrix}$$

$$v_1 = [2 \quad -2] \times \begin{bmatrix} -2 \\ 3 \end{bmatrix} - [2 \quad -2] \times \begin{bmatrix} 2 \\ -2 \end{bmatrix} = -18$$

$$v_2 = [0 \quad 3] \times \begin{bmatrix} -2 \\ 3 \end{bmatrix} - [0 \quad 3] \times \begin{bmatrix} 0 \\ 3 \end{bmatrix} = 0$$

$$v_3 = [-4 \quad 2] \times \begin{bmatrix} -2 \\ 3 \end{bmatrix} - [-4 \quad 2] \times \begin{bmatrix} -4 \\ 2 \end{bmatrix} = -6$$

i.e.,

$$Z^{(1)} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \qquad \text{and} \qquad V^{(1)} = \begin{bmatrix} -18 \\ 0 \\ -6 \end{bmatrix}.$$

Two elements of $Z^{(1)}$ have a different sign from the corresponding elements in $V^{(1)}$. Thus, the algorithm iterates through the elements in $V^{(1)}$ to search for the index of the element $v_i \in V^{(1)}$ with the largest absolute value, such that $z_i \in Z^{(1)}$ and v_i have different signs. This search returns $pos = 1$. In the second iteration, we change the sign of the element at position 1 in $Z^{(1)}$ and we use the new sign vector $Z^{(2)}$ to compute $V^{(2)}$, similar to iteration 1, and get

$$Z^{(2)} = \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix} \text{ and } \quad V^{(2)} = \begin{bmatrix} -18 \\ 12 \\ 18 \end{bmatrix}.$$

Since all corresponding elements in $Z^{(2)}$ and $V^{(2)}$ have the same sign, the algorithm terminates and $Z^{(2)}$ is returned as the maximizing sign vector that maximizes $Z^T \cdot V$.

V. Properties of the Algorithm

This section works out the main properties of the SSV algorithm. More specifically, we prove monotonicity, termination and correctness of our algorithm.

A. Monotonicity

Let $Z^{(k)}$ and $V^{(k)}$ refer, respectively, to vectors V and Z in the k-th iteration of the SSV algorithm. $v_i^{(k)}$ and $z_i^{(k)}$ denote, respectively, the i-th element of $V^{(k)}$ and $Z^{(k)}$. Lemma 4 shows that the computation of the maximizing sign vector in the SSV algorithm is *strictly monotonic*, i.e., each iteration increases the value of $Z^T \cdot V$.

Lemma 4 (Monotonicity): For any two consecutive iterations k and $k+1$ in the SSV algorithm the following holds:

$$(Z^{(k+1)})^T \cdot V^{(k+1)} > (Z^{(k)})^T \cdot V^{(k)}.$$

Proof: First, $(Z^{(k+1)})^T \cdot V^{(k+1)} > (Z^{(k+1)})^T \cdot V^{(k)}$ is proven. Let i be the index of the largest $|v_i^{(k)}|$ such that $v_i^{(k)} \times z_i^{(k)} < 0$. Thus, we change the sign of $z_i^{(k)}$ and compute $V^{(k+1)} = diag^{=0}(\mathbf{X} \cdot \mathbf{X}^T) \cdot Z^{(k+1)}$. For the computation of $v_i^{(k+1)}$, we multiply $z_i^{(k+1)}$ with the i-th diagonal element of $\mathbf{X} \cdot \mathbf{X}^T$. Since all diagonal elements are equal to 0 we get $v_i^{(k)} = v_i^{(k+1)}$. Next, assume a unit vector U_i with the same length as $Z^{(k)}$ where the i-th element is 1 and all other elements are 0. Using U_i we compute $Z^{(k+1)}$ as follows:

$$\begin{aligned} Z^{(k+1)} &= Z^{(k)} - 2 \times U_i \\ diag^{=0}(\mathbf{X} \cdot \mathbf{X}^T) \cdot Z^{(k+1)} &= diag^{=0}(\mathbf{X} \cdot \mathbf{X}^T) \cdot (Z^{(k)} - 2 \times U_i) \\ diag^{=0}(\mathbf{X} \cdot \mathbf{X}^T) \cdot Z^{(k+1)} &= diag^{=0}(\mathbf{X} \cdot \mathbf{X}^T) \cdot Z^{(k)} - \\ & \qquad 2 \times diag^{=0}(\mathbf{X} \cdot \mathbf{X}^T) \cdot U_i \end{aligned}$$

We substitute $V^{(k+1)} = diag^{=0}(\mathbf{X} \cdot \mathbf{X}^T) \cdot Z^{(k+1)}$ and get:

$$\begin{aligned} V^{(k+1)} &= V^{(k)} - 2 \times diag^{=0}(\mathbf{X} \cdot \mathbf{X}^T) \cdot U_i \\ (Z^{(k+1)})^T \cdot V^{(k+1)} &= (Z^{(k+1)})^T \cdot V^{(k)} - \\ & \qquad 2 \times (Z^{(k+1)})^T \cdot diag^{=0}(\mathbf{X} \cdot \mathbf{X}^T) \cdot U_i \end{aligned} \tag{2}$$

Let $Y = (Z^{(k+1)})^T \cdot diag^{=0}(\mathbf{X} \cdot \mathbf{X}^T)$. Since we changed the sign of $z_i^{(k)}$ we have $z_i^{(k+1)} < 0$ and get $y_i < 0$. Since u_i is the only element in U_i that is equal to 1 we know that in $Y \cdot U_i$ only y_i is multiplied by 1, whereas all other elements of Y are multiplied by 0. We use $Y \cdot U_i < 0$ to get $2 \times (Z^{(k+1)})^T \cdot diag^{=0}(\mathbf{X} \cdot \mathbf{X}^T) \cdot U_i < 0$. From (2), we get

$$(Z^{(k+1)})^T \cdot V^{(k+1)} \quad > \quad (Z^{(k+1)})^T \cdot V^{(k)}. \tag{3}$$

Second, we show $(Z^{(k+1)})^T \cdot V^{(k)} > (Z^{(k)})^T \cdot V^{(k)}$. By changing the sign of the element in $Z^{(k)}$ that corresponds to the largest $|v_i^{(k)}|$ such that $v_i^{(k)} \times z_i^{(k)} < 0$, we get:

$$\begin{aligned} \sum_{i=1}^{n} (z_i^{(k+1)} \times v_i^{(k)}) \quad &> \quad \sum_{i=1}^{n} (z_i^{(k)} \times v_i^{(k)}) \\ (Z^{(k+1)})^T \cdot V^{(k)} \quad &> \quad (Z^{(k)})^T \cdot V^{(k)} \end{aligned} \tag{4}$$

By transitivity using (3) and (4), the following holds:

$$(Z^{(k+1)})^T \cdot V^{(k+1)} > (Z^{(k)})^T \cdot V^{(k)}. \qquad \blacksquare$$

B. Termination

Lemma 5 (Termination): Let $\mathbf{X}$ be an $n \times m$ matrix. SSV($\mathbf{X}, n, m$) terminates and performs at most n iterations.

Proof: We show that each element in the sign vector, $z_i \in Z$, is changed at most once. Since Z contains n elements, the algorithm performs at most n iterations.

To prove that each element in the sign vector is changed at most once, we show that any value $v_i \in V$ that increases $Z^T \cdot V$ most in one of the iterations does not change its sign in subsequent iterations, i.e., $v_i < 0$. This prevents v_i to be considered as candidate element in future iterations, since the

corresponding sign z_i is changed from 1 to -1 and has the same sign as v_i. From Lemma 1 we have

$$V^{(k)} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & x_{12} & \dots & x_{1n} \\ x_{21} & 0 & \dots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & 0 \end{bmatrix}}_{diag^{=0}(\mathbf{X}\cdot\mathbf{X}^T)} \cdot Z^{(k)}$$

We show two consecutive iterations, $k+1$ and $k+2$, and assume without loss of generality that z_1 and z_n are changed, respectively.

Let $v_1^{(k)}$ be the element that increases $Z^T \cdot V$ most at the k-th iteration and assume that $z_n^{(k)} = 1$ has not yet been changed. We have

$$v_1^{(k)} = (z_2^{(k)} \times x_{12}) + \ldots + (z_{n-1}^{(k)} \times x_{1n-1}) + x_{1n} < 0. \quad (5)$$

Next, consider iteration $k+1$. Let $v_i^{(k+1)} < 0$ with $i \neq 1$ be the element that increases $Z^T \cdot V$ most (if no such element exists the algorithm terminates). Without loss of generality assume that $i = n$. Then, the sign of z_n is changed from 1 to -1, i.e., $z_n^{(k+2)} = -1$. Hence, in iteration $k+2$ we get

$$v_1^{(k+2)} = (z_2^{(k+2)} \times x_{12}) + \ldots + (z_{n-1}^{(z+2)} \times x_{1n-1}) - x_{1n}. \quad (6)$$

Now, we perform a case analysis on the element x_{1n} and prove that $v_1^{(k+2)} < 0$ holds.

a) Case $x_{1n} \geq 0$: We have that $z_i^{(k)} = z_i^{(k+2)}$ for $i = 2, \ldots, n-1$, thus the sum over the first $n-1$ elements in $v_1^{(k)}$ (equation 5) and $v_1^{(k+1)}$ (equation 6) is the same. Since $x_{1n} \geq 0$ we can conclude that $v_1^{(k+2)} < v_1^{(k)} < 0$.

b) Case $x_{1n} < 0$: As in the previous case, let $v_1^{(k)}$ and $v_n^{(k+1)}$ be the two elements that increase $Z^T \cdot V$ most at iteration k and $k+1$, respectively. Since $diag^{=0}(\mathbf{X}\cdot\mathbf{X}^T)$ is a symmetric matrix, we have $x_{1n} = x_{n1}$. Again, we do a case analysis on x_{1n} (for simplicity, we omit z elements in the following equations):

- *Case* $x_{1n} < x_{n2} + \ldots + x_{nn-1}$: By simple transformations and $x_{1n} = x_{n1}$ we get

  $$0 < -x_{n1} + x_{n2} + \ldots + x_{nn-1} = v_n^{(k+1)}.$$

  This leads to a contradiction with our assumption that $v_n^{(k+1)}$ increases $Z^T \cdot V$ most in iteration $k+1$, hence $v_n^{(k+1)} < 0$.

- *Case* $x_{1n} \geq x_{n2} + \ldots + x_{nn-1}$: Since $v_1^{(k)}$ increases $Z^T \cdot V$ most at the k-th iteration, the following holds:

  $$v_1^{(k)} < v_n^{(k)}$$
  $$x_{12} + \ldots + x_{1n-1} + x_{1n} < x_{1n} + x_{n2} \ldots + x_{nn-1}$$
  $$x_{12} + \ldots + x_{1n-1} < x_{n2} + \ldots + x_{n-1n}$$

  We substitute the right-hand side in the above equation by our assumption and get $x_{12} + \ldots + x_{1n-1} < x_{1n}$ and further $x_{12} + \ldots + x_{1n-1} - x_{1n} = v_1^{(k+2)} < 0$.

By using a similar reasoning, we can generalize the proof for iterations k and $k+p$ with $1 \leq p < n-k$ and elements v_i and v_j, $i \neq j$, that increase $Z^T \cdot V$ most, respectively. ■

C. Greedy Strategy

Lemma 6 (Local optimal choice): The SSV algorithm changes in each iteration the element of the sign vector Z that most increases $Z^T \cdot V$.

Proof (idea): The SSV algorithm changes in each iteration the sign of an element $z_i^{(k)}$ for which $z_i^{(k)} \times v_i^{(k)} < 0$. This strategy increases $Z^T \cdot V$. If instead we would change the sign of an element $z_i^{(k)}$ for which $z_i^{(k)} \times v_i^{(k)} \geq 0$ we get $Z^{(k+1)} = Z^{(k)} + 2 \times U$ (cf. Lemma 4), which implies $(Z^{(k+1)})^T \cdot V^{(k+1)} \leq (Z^{(k)})^T \cdot V^{(k)}$. In the algorithm we go through all elements for which $z_i^{(k)} \times v_i^{(k)} < 0$ and choose the element that maximizes $|z_i^{(k)} \times v_i^{(k)}|$. Since this also holds for $k+1$, then we make the local optimal choice. ■

D. Correctness

In this section we prove that our greedy approach computes the optimal solution.

Lemma 7 (Global maximum): The SSV algorithm computes the maximizing sign vector for which the final product $Z^T \cdot V$ is globally maximal.

Proof: In order to prove the correctness of our greedy algorithm, we need to demonstrate that our algorithm satisfies two properties that make any greedy approach optimal (see [31] for further details).

1) Greedy Choice Property: This property states that an optimal solution exists that is consistent with the first greedy choice. We demonstrate that there exists an optimal solution which includes the first greedy choice.

Let $Z = [z_1, \ldots, z_n]$ be a sign vector and $P = \{p_1, \ldots, p_k\}$, $k \leq n$, be the ordered set of sign change positions in Z as computed by the SSV algorithm. For instance, $P = \{2, 3, 4, 1\}$ indicates that in the first iteration z_2 has been changed, in the second iteration z_3, etc. We use Z_P to refer to the sign vector where the positions in P have been flipped. Furthermore, let $P^* = \{p_1^*, \ldots, p_l^*\}$ be the ordered set of sign change positions in Z for the optimal solution. We can distinguish two cases:

- $p_1 \in P^*$: The first greedy choice is included in the optimal solution, hence the greedy choice property holds.
- $p_1 \notin P^*$: The first greedy choice is not included in the optimal solution. Without loss of generality we substitute the first element $p_1^* \in P^*$ by p_1 and get $P' = (P^* - \{p_1^*\}) \cup \{p_1\} = \{p_1, p_2^*, \ldots, p_n^*\}$. From the greedy strategy we know that $z_{p_1} \times v_{p_1} < 0$ and that p_1 is the position with the largest value, i.e., $|v_{p_1}^{(k)}| \geq |v_i^{(k)}|$ for $i = 1, \ldots, n$. Therefore, by replacing p_1^* with p_1 in P' the product $Z_{P'}^T \cdot V$ will increase, which is a contradiction to the fact that P^* are the positions of the optimal solution.

Thus, the first greedy choice is part of the optimal solution.

2) Optimal Substructure Property: This property states that solutions to subproblems of an optimal solution are also optimal. We demonstrate by contradiction that the optimal

solution after a greedy choice contains an optimal solution to the remaining subproblem.

Let $P^* = \{p_1^*, \ldots, p_k^*\}$ be the ordered set of sign change positions in Z for the optimal solution and let $P' = P^* - \{p_1^*\} = \{p_2^*, \ldots, p_k^*\}$ be the ordered set of sign changes of all positions but p_1^*. Assume that P' is not optimal. Then, there exists an optimal solution P'' with $|P''| = k-1$ and $p_1^* \notin P''$ such that

$$(Z_{P''})^T \cdot V > (Z_{P'})^T \cdot V.$$

By adding the position p_1^* on both sides we obtain

$$(Z_{P'' \cup \{p_1^*\}})^T \cdot V > (Z_{P' \cup \{p_1^*\}})^T \cdot V = (Z_{P^*})^T \cdot V.$$

This contradicts our assumption that P^* produces the optimal solution. Therefore, P' is optimal.

Since the SSV algorithm satisfies the greedy choice property and the optimal substructure property, we conclude that the result of the algorithm is a global optimum. ■

E. Complexity Analysis

The SSV algorithm keeps in memory the sign vector Z and V, each with $O(n)$ space complexity, where n is the number of rows in $\mathbf{X}$. Therefore, the total space complexity is $O(n)$.

The total runtime complexity is $O(xn)$, where x is the number of changed elements in the returned sign vector Z. In the worst case, the sign of each element is changed, yielding a time complexity of $O(n^2)$. The experiment in Fig. 10 shows that the average number of sign changes in Z is $\frac{n}{2}$.

VI. Empirical Evaluation

A. Setup

We refer to SCD and QCD as the Centroid Decomposition (CD) using respectively SSV and QSV. We implemented SCD, QCD and SVD algorithms in Java on top of an Oracle database. We connect to the database through the 11.2 JDBC driver. For the experiments the client and the database server run on the same 2.6 GHz machine with 4GB RAM.

The empirical evaluation is performed on real world datasets that describe hydrological time series[1] where each tuple records a timestamp and a value of a specific observation. The hydrological time series have been normalized with the z-score normalization technique [32]. The values of the observations are stored as 4-byte floating numbers. We conducted also experiments on raw time series from the UCR repository [33].

In what follows, we evaluate scalability, efficiency and correctness of our algorithm. Furthermore, we empirically determine the number of iterations performed by the SSV algorithm and we show the impact of the distribution of the sign of values across different time series on the number of iterations. For each experiment, we display the average result over five runs of the algorithms.

[1]The data was kindly provided by the environmental engineering company HydroloGIS (http://www.hydrologis.edu).

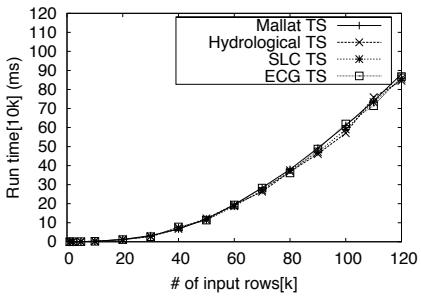
B. Experiments

1) Efficiency and Scalability: In order to evaluate the efficiency and scalability, we choose the longest time series from the UCR repository. We concatenate the time series that belong to the same dataset to get time series with the same length as the hydrological ones. Table I describes the used time series.

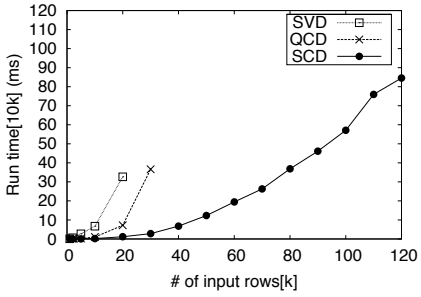
TABLE I. Description of first set of time series.

Name	Provenance	Max_Length	Number TS
Hydrological TS	Hydrologis	120'000	217
MALLAT	UCR repository	2345	1024
StarLightCurves (SLC)	UCR repository	8236	1024
CinC_ECG_torso (ECG)	UCR repository	1380	1639

The experiment in Fig. 5 evaluates the runtime of the SCD algorithm and compares it against other techniques. The computation of matrix $\mathbf{X} \cdot \mathbf{X}^T$ is included in the running times of QCD and SVD. In Fig. 5(a), the number m of time series is four and the number n of rows varies between zero and $120k$. This experiment shows that, for all time series, SCD has quadratic runtime with respect to the number of rows of the input matrix $\mathbf{X}$. Fig. 5(b) compares the runtime of SCD against QCD and SVD using hydrological time series. This experiment shows that SCD, QCD and SVD have quadratic runtime. The QCD algorithm runs out of memory for $n > 30k$, whereas SVD runs out of memory for $n > 20k$. In contrast, SCD performs the decomposition of a matrix that contains four time series of $120k$ observations each in less than seven minutes.



(a) Runtime of SCD using different time series



(b) Runtime of different techniques using hydrological time series

Fig. 5. Runtime by varying n.

In the experiment in Fig. 6, n is set to $5k$ and m varies between 20 and 100. The results show that the runtime of the SCD and QCD algorithms increases linearly with m. Using SCD, the decomposition of a matrix of 100 time series

with $5k$ observations each is performed in approximately 80 seconds. We did not include SVD, which has a cubic runtime complexity with respect to m.

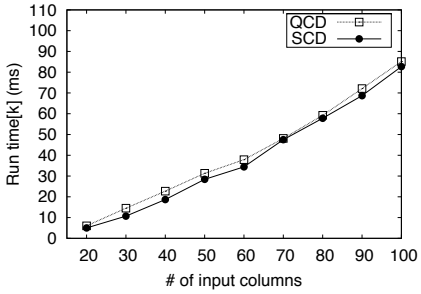


Fig. 6. Runtime by varying m in Hydrological time series.

Fig. 7 compares the memory usage of SCD against QCD and SVD (notice the log-scale on the y-axis). For each of the three algorithms we sum the allocated space for all data structures. The results of the calculation of memory allocation confirm the linear space complexity of SCD with respect to the number of rows of the input matrix $\mathbf{X}$, whereas QCD and SVD have quadratic space complexity. For $n > 30k$ and $n > 20k$, respectively, QCD and SVD run out of memory.

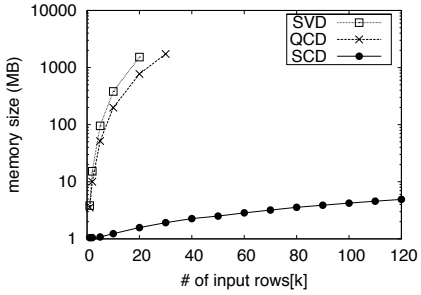


Fig. 7. Memory usage.

2) Algorithm Properties: The properties of the SSV algorithm are evaluated using the time series described in Table I.

Fig. 8 evaluates the trend of the product $(Z^{(k)})^T \cdot V^{(k)}$ computed by the SSV algorithm. This experiment confirms the monotonicity property stated in Lemma 4. We extract 1000 values from each time series and compute the product $(Z^{(k)})^T \cdot V^{(k)}$ for 20 iterations. The experiment shows that $(Z^{(k)})^T \cdot V^{(k)}$ computed by our algorithm is monotonically increasing.

In Fig. 9, we compare the sign vectors computed by the SSV algorithm against those computed by QSV algorithm. This experiment aims to confirm the correctness property stated in Lemma 7. We compute the percentage of correct sign vectors, i.e., the sign vectors computed by SSV that are equal to those computed by QSV. As expected, the experiment confirms that SSV computes the correct sign vectors in all cases.

3) Number of Iterations: The time series from Table I are used. In the experiment of Fig. 10, we show the number of iterations in the SSV algorithm that are required to compute the maximizing sign vectors. For all used time series, our

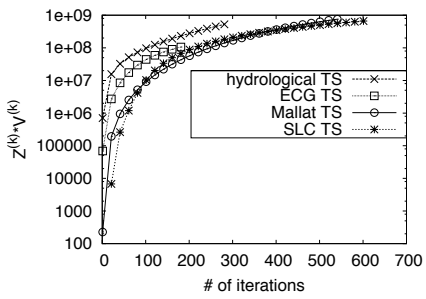


Fig. 8. Monotonicity property of SSV.

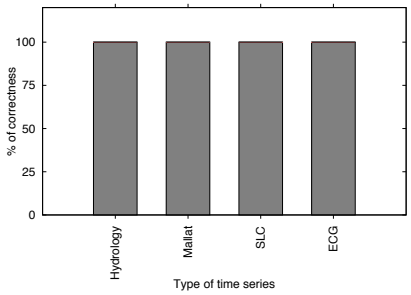


Fig. 9. Correctness of SSV.

algorithm performs on average $\frac{n}{2}$ iterations, i.e., it performs only half of the maximum number of iterations.

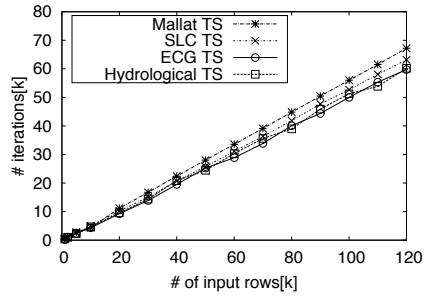


Fig. 10. Number of iterations in SSV.

4) Impact of Sign of Values: We select three time series from the UCR repository in such a way that we have different distributions of negative values at the same timestamp across the selected time series. Table II describes the used time series, where the third column is the number of values with a negative sign at the same timestamp in all time series.

TABLE II. DESCRIPTION OF SECOND SET OF TIME SERIES.

Name	Provenance	length	# negative rows (x)
Gun_Point	UCR	150	90
Cricket_X	UCR	300	21
Beef	UCR	470	0

Fig. 11 illustrates the impact of the number of negative rows (x) on the number of iterations, and hence on the runtime. In the Gun_Point dataset, x is bigger than half of the input rows. The number of iterations is on average equal to half of the input rows. In the Cricket_X dataset, x is between 1 and half of the input rows. In this case, the SSV algorithm iterates

on average $x+1$ times. In the case where all values in all time series have the same sign, the number of iterations is equal to 1 as expected. That is, if the sign of all elements is positive or negative, all elements of the weight vector computed in the first iteration of the algorithm, i.e., $V^{(1)}$, are positive. In both cases the sign vector that contains only 1s is the maximizing vector and our algorithm requires only one iteration to find the maximizing vector. Fig. 11 shows also that the increase in the number of input rows together with a higher x implies a higher runtime. In the case where the negative sign is randomly distributed across different time series, the number of iterations is on average equal to $\frac{n}{2}$ as shown in Fig. 10.

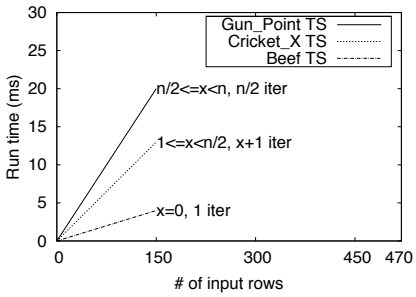


Fig. 11. Impact of sign of input rows.

In Fig. 12, we used the Beef time series that does not contain any negative value and we incrementally change the sign of 10% of input rows. This experiment shows that the runtime increases with the number of negative rows. If all rows are negative, the runtime is the same as when all rows are positive and the number of iterations performed by SSV is equal to 1.

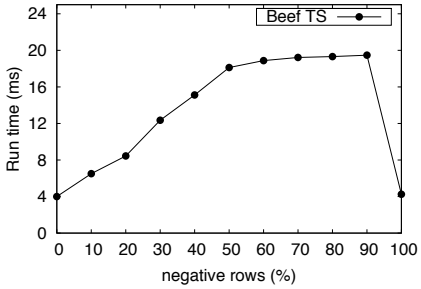
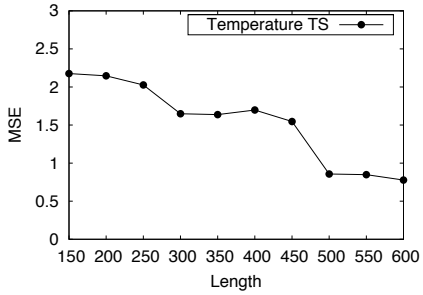


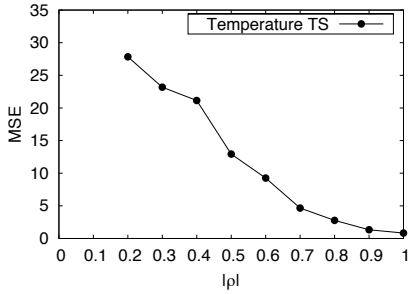
Fig. 12. Run time with varying percentage of negative rows.

5) Length of Time Series: The final experiment shows the impact of using long time series for the recovery of missing values [15]. We remove a block of 100 values from a temperature time series and recover the removed block by an iterative computation of matrices $\mathbf{L}$ and $\mathbf{R}$. The input matrix contains as columns the time series with the removed block and three other temperature time series. Then, we perform a one rank reduction, i.e., we compute only three vectors in the matrices $\mathbf{L}$ and $\mathbf{R}$ instead of four, and we iterate until the difference in the Frobenius norm [34] between the matrix before the decomposition and the one after the decomposition is less than 10^{-5}. To measure the accuracy, we compute the Mean Square Error ($MSE = \frac{1}{k}\sum_{i=1}^{k}(\tilde{x}_i - x_i)^2$; original value x_i; recovered value $\tilde{x}_i$; number of observations k) between the original and the recovered blocks [35], [36].

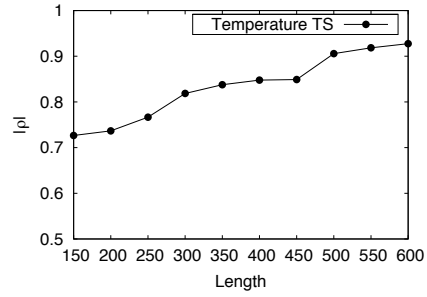
Fig. 13 shows the result of this experiment. In Fig. 13(a), the length of the time series is varied. Longer time series significantly reduce the MSE. In Fig. 13(b), we take different



(a) Length of TS vs MSE



(b) Correlation vs MSE



(c) length of TS vs Correlation

Fig. 13. Impact of the length of the time series and the correlation between them on the recovery of missing values.

time series of 200 values such that the absolute value of the Pearson correlation ρ between the time series with the missing block and the other time series varies, and we compute the MSE. The experiment shows that the more correlated the time series are the better is the recovery. Finally, Fig. 13(c) shows that the correlation between the time series increases with the length of time series (the same time series as in Fig. 13(a) are used).

VII. Conclusion and Future Work

In this paper, we introduced the Scalable Sign Vector algorithm that performs the Centroid Decomposition of a matrix in linear space complexity. We provided proofs that show the scalability, the termination and the correctness of our algorithm. An empirical evaluation on real world hydrological data sets and also on data sets from the UCR repository demonstrates that our algorithm has the same runtime as the most

efficient algorithm to compute the Centroid Decomposition, but reduces the space complexity from quadratic to linear.

In future work, we plan to investigate an incremental version of the Centroid Decomposition that could be applied for dynamic time series. Another promising direction is to investigate segmentation techniques of time series.

ACKNOWLEDGMENTS

The first author is supported by a grant from Hasler Foundation in Switzerland. The authors would like to thank HydroloGIS [37] for providing the hydrological datasets that we have used to conduct our experiments, and Eszter Börzsönyi for her contribution to the implementation part of this work. We would also like to give special thanks to the anonymous reviewers for their insightful comments.

REFERENCES

[1] K. Karadimitriou and J. M. Tyler, "The centroid method for compressing sets of similar images," *Pattern Recognition Letters*, vol. 19, no. 7, pp. 585 – 593, 1998.

[2] M. Chu and R. Funderlic, "The centroid decomposition: Relationships between discrete variational decompositions and svds," *SIAM J. Matrix Anal. Appl.*, vol. 23, no. 4, pp. 1025–1044, 2001.

[3] W. Lang, M. Morse, and J. Patel, "Dictionary-based compression for long time-series similarity," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 22, no. 11, pp. 1609–1622, 2010.

[4] M. Chu, R. Funderlic, and G. Golub, "A rank-one reduction formula and its applications to matrix factorizations," *SIAM Review*, vol. 37, no. 4, pp. 512–530, 1995.

[5] I. Dhillon and D. Modha, "Concept decompositions for large sparse text data using clustering," *Mach. Learn.*, vol. 42, no. 1-2, pp. 143–175, 2001.

[6] H. Park, L. Jeon, and J. Rosen, "Lower dimensional representation of text data based on centroids and least squares," *BIT*, vol. 43, pp. 427–448, 2003.

[7] U. Rebbapragada, P. Protopapas, C. Brodley, and C. Alcock, "Finding anomalous periodic time series," *Mach. Learn.*, vol. 74, no. 3, pp. 281–313, 2009.

[8] G. Golub and C. Van Loan, *Matrix computations (3rd ed.)*. Baltimore, MD, USA: Johns Hopkins University Press, 1996.

[9] C. Meyer, Ed., *Matrix analysis and applied linear algebra*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2000.

[10] D. Skillicorn, *Understanding Complex Datasets: Data Mining with Matrix Decompositions (Chapman & Hall/Crc Data Mining and Knowledge Discovery Series)*. Chapman & Hall/CRC, 2007.

[11] K. V. Ravi Kanth, D. Agrawal, and A. Singh, "Dimensionality reduction for similarity searching in dynamic databases," in *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '98. New York, NY, USA: ACM, 1998, pp. 166–176.

[12] J. Han, *Data Mining: Concepts and Techniques*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005.

[13] J. Ye, R. Janardan, and Q. Li, "Gpca: an efficient dimension reduction scheme for image compression and retrieval," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '04. New York, NY, USA: ACM, 2004, pp. 354–363.

[14] P. Waldemar and T. Ramstad, "Hybrid klt-svd image compression," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, ser. ICASSP '97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 2713–2716.

[15] M. Khayati and M. Böhlen, "Rebom: Recovery of blocks of missing values in time series," in *Proceedings of the 2012 ACM International Conference on Management of Data*, ser. COMAD '12. Computer Society of India, 2012, pp. 44–55.

[16] M. Brand, "Fast online svd revisions for lightweight recommender systems," in *SDM*, 2003.

[17] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*. SIAM: Society for Industrial and Applied Mathematics, 1997.

[18] S. Rendle., "Factorization machines," in *ICDM*, 2010, pp. 995–1000.

[19] S. Rendle, "Scaling factorization machines to relational data," in *Proceedings of the 39th international conference on Very Large Data Bases*, ser. PVLDB'13. VLDB Endowment, 2013, pp. 337–348.

[20] E. Papalexakis, C. Faloutsos, and N. Sidiropoulos, "Parcube: Sparse parallelizable tensor decompositions," in *ECML/PKDD (1)*, 2012, pp. 521–536.

[21] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009.

[22] R. Gemulla, E. Nijkamp, P. Haas, and Y. Sismanis, "Large-scale matrix factorization with distributed stochastic gradient descent," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '11. New York, NY, USA: ACM, 2011, pp. 69–77.

[23] M. Zinkevich, M. Weimer, A. J. Smola, and L. Li, "Parallelized stochastic gradient descent," in *NIPS*, 2010, pp. 2595–2603.

[24] B. Li, S. Tata, and Y. Sismanis, "Sparkler: supporting large-scale matrix factorization," in *Proceedings of the 16th International Conference on Extending Database Technology*, ser. EDBT '13. New York, NY, USA: ACM, 2013, pp. 625–636.

[25] R. S. Xin, J. Rosen, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica, "Shark: Sql and rich analytics at scale," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '13. New York, NY, USA: ACM, 2013, pp. 13–24.

[26] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[27] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," *IEEE Trans. on Knowl. and Data Eng.*, vol. 17, no. 6, pp. 734–749, 2005.

[28] F. McSherry and I. Mironov, "Differentially private recommender systems: Building privacy into the netflix prize contenders," in *KDD*, 2009, pp. 627–636.

[29] J. Bennett and S. Lanning, "The netflix prize," *KDD cup and workshop*, 2007.

[30] B. N. Miller, I. Albert, S. K. Lam, J. A. Konstan, and J. Riedl, "Movielens unplugged: experiences with an occasionally connected recommender system," in *Proceedings of the 8th international conference on Intelligent user interfaces*, ser. IUI '03. New York, NY, USA: ACM, 2003, pp. 263–266.

[31] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, 3rd ed. The MIT Press, 2009.

[32] A. Jain, K. Nandakumar, and A. Ross, "Score normalization in multimodal biometric systems," *Pattern Recognition*, vol. 38, no. 12, pp. 2270–2285, 2005.

[33] E. Keogh, Q. Zhu, B. Hu, H. Y., X. Xi, L. Wei, and C. Ratanamahatana, The UCR Time Series Classification/Clustering, homepage: www.cs.ucr.edu/~eamonn/time_series_data/, 2011.

[34] C. Ma, Y. Kamp, and L. Willems, "A frobenius norm approach to glottal closure detection from the speech signal," *Speech and Audio Processing, IEEE Transactions on*, vol. 2, no. 2, pp. 258–265, 1994.

[35] L. Li, J. McCann, N. S. Pollard, and C. Faloutsos, "Dynammo: mining and summarization of coevolving sequences with missing values," in *KDD*, 2009, pp. 507–516.

[36] F. Kahl, A. Heyden, and L. Quan, "Minimal projective reconstruction including missing data," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 23, no. 4, pp. 418–424, 2001.

[37] HydroloGIS, http://www.hydrologis.eu/, accessed September 14, 2012.