RESULTS reproduced

ORBITS: Online Recovery of Missing Values in Multiple Time Series Streams

Mourad Khayati, Ines Arous, Zakhar Tymchenko, Philippe Cudré-Mauroux
University of Fribourg
Switzerland
{firstname.lastname}@unifr.ch

ABSTRACT

With the emergence of the Internet of Things (IoT), time series streams have become ubiquitous in our daily life. Recording such data is rarely a perfect process, as sensor failures frequently occur, yielding occasional blocks of data that go missing in multiple time series. These missing blocks do not only affect real-time monitoring but also compromise the quality of online data analyses. Effective streaming recovery (imputation) techniques either have a quadratic runtime complexity, which is infeasible for any moderately sized data, or cannot recover more than one time series at a time.

In this paper, we introduce a new online recovery technique to recover multiple time series streams in linear time. Our recovery technique implements a novel incremental version of the Centroid Decomposition technique and reduces its complexity from quadratic to linear. Using this incremental technique, missing blocks are efficiently recovered in a continuous manner based on previous recoveries. We formally prove the correctness of our new incremental computation, which yields an accurate recovery. Our experimental results on real-world time series show that our recovery technique is, on average, 30% more accurate than the state of the art while being vastly more efficient.

PVLDB Reference Format:
Mourad Khayati, Ines Arous, Zakhar Tymchenko, Philippe Cudré-Mauroux. ORBITS: Online Recovery of Missing Values in Multiple Time Series Streams. PVLDB, 14(3): 294 - 306, 2021.
doi:10.14778/3430915.3430920

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/eXascaleInfolab/orbits.

1 INTRODUCTION

Time series are ubiquitous in many domains, e.g., finance, hydrology, network monitoring, or the Internet of Things (IoT). In such applications, time series often contain a large number of missing values that occur as blocks of consecutive values because of sensor failure, power outages, transmission problems, etc. [4, 37]. Data management systems assume no such gaps exist in the data. Even if a system can work with incomplete data (e.g., NULLs in databases), leaving missing values untreated can cause incorrect or ill-defined results [7, 9]. Missing values can alter time series statistical properties, such as the correlation. This in turn, might significantly affect further analysis tasks, e.g., data sampling, or exploration, rendering their results pointless [14]. Moreover, repairing dirty series substantially improves downstream tasks such as classification [39].

The recovery of these missing blocks is challenging, as in many modern applications data arrives in a streaming fashion. The stream could be infinite, rendering batch recovery techniques that attempt to process the entire stream for analysis impractical [22]. In many mission-critical applications, the missing blocks need to be recovered as they are encountered. We describe below two real-world applications that require the continuous recovery of missing blocks in time series streams. We empirically evaluate these two applications in Section 6.

Example 1. We consider the example of an IoT-based sports analytical use-case, where millions of sensors are unobtrusively integrated into the clothing and equipment of players to monitor their activities in real-time [13]. The monitoring aims to examine the players' performance and minimize the risk of injury during practice. Missing blocks occur here as the sensors often detach from the players. Recovering these holes on-the-fly allows coaches to immediately adjust their tactics by repositioning the players on the field, which in turn might improve the overall performance of the team.

Example 2. Consider as a second example the case of the Swiss Federal Office for the ENvironment (FOEN) [1], an organization responsible for protecting against natural hazards and triggering alerts in case of floods in Swiss lakes and rivers. FOEN operates more than 260 water stations spread all over the country, each continuously collecting water discharge measurements. These water stations frequently suffer sensor failures or errors in the transmission of the data leading to missing blocks. However, the continuous monitoring of the current water level is crucial to swiftly warn about imminent threats such as floods.

Online recovery of missing values is a well-studied problem and a number of algorithms have been proposed in the past to tackle it. There is, however, a lack of practical techniques able to efficiently recover multiple incomplete streams by achieving real-time complexity, which is an essential requirement for any streaming algorithm. Existing online algorithms can be classified into two classes, according to the underlying method they use. On one hand, matrix completion techniques operate by constructing a matrix of series and then applying a low-rank reduction of this matrix to derive the principal dimensions that represent the streams. Each time the stream is updated, an approximated reduced matrix is computed out of the previous one, which limits the accuracy of these techniques. On the other hand, pattern-based techniques

Proceedings of the VLDB Endowment, Vol. 14, No. 3 ISSN 2150-8097.
doi:10.14778/3430915.3430920

apply similarity techniques to detect repeating trends and use them to derive replacements for the missing blocks. These algorithms operate in an online manner by evaluating dependencies across streams. The calculation of the exact set of dependencies requires a large number of iterations rendering the recovery inefficient.

In practice, time series streams often exhibit temporal and spatial similarity that can be used to summarize the main properties of the data. We leverage this observation in our work by incrementally computing a handful of dimensions, which conceptually represent the up-to-date streams, and use them to recover missing values in a continuous fashion. The proposed online recovery efficiently implements an *exact* incremental version of the Centroid Decomposition (CD) technique. It achieves an accurate recovery thanks to the ability of CD to embed the correlation across time series as part of its decomposition process [19, 21]. Our incremental approach can be beneficial to various applications where CD has been applied such as factor analysis [10], text classification [15], image compression [17], or anomaly detection [23].

At a technical level, CD decomposes an input matrix $\mathbf{X}$ into the product of two matrices, $\mathbf{X} = \mathbf{L} \cdot \mathbf{R}^T$, where $\mathbf{L}$ and $\mathbf{R}$ are called the loading and relevance matrix, respectively. The most expensive operation in CD is the computation of maximizing sign vectors, Z, which consist of 1s and -1s and are used as part of an optimization problem that must be solved to obtain the decomposition. The only online CD approach, called UpdateCD [35], recomputes the sign vectors from scratch each time new data arrives, thus achieving the same *quadratic* time complexity as the batch algorithm [20].

In this paper, we propose to vastly improve the efficiency of the Centroid Decomposition technique. Our algorithm extends the traditional iterative way of calculating the sign vectors using two techniques. First, it uses an anticipatory technique that speeds up the CD computation by predicting the termination of the sign vector search and thus reduces the number of iterations. Second, it utilizes the matrix similarity before and after new data arrives to compute the sign vectors incrementally.

In summary, the main contributions of this paper are as follows:

- We introduce a new algorithm called ORBITS for the Online Recovery of missing Blocks In multiple Time Series streams. ORBITS relies on an anticipatory computation of the CD technique, which reduces its time complexity from quadratic to linear (Section 4).
- We prove the dependence between sign vectors before and after each stream update. We use this property to prove the correctness of our decomposition, which guarantees an accurate recovery of missing values (Section 5).
- We evaluate ORBITS on various time series datasets and show that our technique substantially outperforms the state of the art both in accuracy and efficiency (Section 6).

2 RELATED WORK

We describe, in turn, related work on (a) streaming recovery techniques and (b) Centroid Decomposition.

2.1 Streaming Recovery Techniques

Online matrix completion algorithms have been used for the continuous recovery of missing values. They assume that the streams present a temporal continuity and apply matrix decomposition/factorization techniques for the recovery.

Incremental SVD (ISVD) [8] is the first online matrix completion technique that was proposed. It continuously recovers a matrix with missing values using the Singular Value Decomposition (SVD) method [34]. ISVD first constructs an orthogonal random matrix $\mathbf{S}$ and uses it to approximate the observed values in $\mathbf{X}$. Then, the columns of $\mathbf{S}$ are appended with the data and decomposed using SVD, recovering missing values in the process. To preserve the orthogonality of the decomposition, ISVD requires column updates, and cannot properly handle row updates, i.e., stream values (which is the main focus of this paper).

SAGE (aka online GROUSE) [5, 40] is an online recovery algorithm that relies on PCA (Principal Component Analysis) [16]. PCA takes an $n \times m$ matrix and finds n principal components vectors, which better represent the dimensions of the initial data. SAGE updates the principal components with the appended matrix rows and applies a Stochastic Gradient Descent (SGD) procedure to derive a new matrix such that their product approximates the input matrix. The resulting matrix is used to replace the missing values in the case of column or row updates. SAGE outperforms ISVD both in accuracy and efficiency on column updates [18]. The gradient process makes SAGE inefficient in long time series, and inaccurate in the presence of outliers or lowly correlated time series.

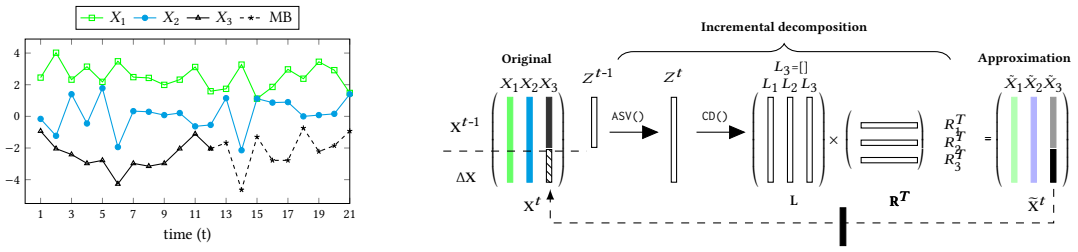
In [27], the authors introduce another online PCA-based recovery technique, which we refer to as pcaMME. It first constructs samples of a covariance matrix using a memory-efficient partitioning technique [26]. The samples are obtained by partitioning the components of the input matrix into separate blocks. Then, pcaMME incrementally derives the largest principal components from the matrix samples, recovering initialized missing values in the process. pcaMME relies on a fast approximation of the principal components, which limits its accuracy.

Online pattern matching techniques have been used in this context as well. These algorithms assume that close sensors can present trend similarity, which is used to derive replacement values.

TKCM [36] identifies and uses repeating patterns (seasonality) in the history of time series to recover missing blocks. It creates a query pattern composed of the most recent measurements and searches for the missing value in the most similar (non-overlapping) pattern to the query pattern. TKCM is able to recover missing values as they appear in time series that exhibit seasonal patterns. The pattern search assumes a single base time series rendering the technique unable to recover more than one series at a time.

OGDImpute [3] uses an auto-regressive (AR) model to recover missing values in data streams. It initializes the missing values with zeros and applies stochastic gradient to compute the AR coefficients. These coefficients are incrementally used to compute two predictions: the first one for the observed values and the second one for the missing values. Such a two-fold prediction allows for learning the trend from the base time series and its similarity to the reference time series. OGDImpute assumes autoregressive input data, which is not the case in most real-world time series.

Yoon et al. [38] introduce a Multi-directional Recurrent Neural Network (MRNN) technique to recover missing blocks. The proposed technique contains an interpolation block and an imputation block, which are simultaneously trained using a fixed size sliding



(a) (b)

Figure 1: Graphical illustration of the recovery of the missing block (MB) in X_3 using ORBITS. (a) A plot of three input time series X_1, X_2 and X_3 with a missing block represented by a dashed line at $13 \leq t \leq 21$. (b) ORBITS's recovery, where CD decomposes the original data, exposing the matrix rank with dimensions L_1, L_2 and L_3.

window. It learns the data dependencies by leveraging both the correlation within time series and the correlation across time series. MRNN resorts to a forward and backward propagation process, which is expensive to compute and can only handle time series that are linearly dependent on one another. Additionally, the performance of the MRNN model deteriorates in short time series because of the limited number of training samples.

Hybrid online techniques that use both matrix completion and pattern matching have also been investigated. For instance, SPIRIT [30, 31] combines PCA with an AR model to reduce a set of co-evolving streams to a small number of hidden variables. These variables summarize the most important features of the original data. For each variable, SPIRIT fits an AR model on historical values and incrementally updates it as new data arrives. Then, this model is used to predict the value of each variable, and subsequently, an estimate of the missing value is derived. The AR model yields an efficient recovery but for a single incomplete time series only.

2.2 Centroid Decomposition (CD)

The *Scalable Sign Vector* (SSV) [20] is the most efficient algorithm to date to compute CD in batch. The main idea of SSV is as follows: instead of searching for the maximizing sign vector using all elements of the input matrix $\mathbf{X}$, SSV searches for it by rows of $\mathbf{X}$. The search is performed by iteratively computing a weight vector V, derived from $\mathbf{X}$, which is then used to select the element in Z that has to be flipped. The computation of the weight vectors is described in detail in Section 3. SSV achieves linear space complexity but has quadratic runtime complexity.

UpdateCD [6] is the only online technique to compute CD. It incrementally decomposes an updated matrix, $\widetilde{\mathbf{X}}$, based on the decomposition before the update, CD($\mathbf{X}$), and the row to be appended. UpdateCD exploits the property that as a result of the decomposition, the loading matrix $\mathbf{L}$ is a stationary point, i.e., CD($\mathbf{L}$) = $\mathbf{L} \times \mathbf{I}^T$ (where $\mathbf{I}$ is the identity matrix). CD($\widetilde{\mathbf{X}}$) is obtained by orthogonal rotation of the loading matrix $\mathbf{L}$. More specifically, the orthogonal rotations are used to construct an intermediate matrix $\mathbf{S}$, from which matrices $\widetilde{\mathbf{L}}$ and $\widetilde{\mathbf{R}}$ of the updated matrix $\widetilde{\mathbf{X}}$ are derived based on CD($\mathbf{S}$). UpdateCD recomputes the sign vectors from scratch each time new data arrives and thus has the same quadratic runtime complexity as SSV.

3 BACKGROUND AND PROBLEM DEFINITION

3.1 Notations

We use bold upper-case letters to refer to matrices, regular font upper-case letters to refer to vectors (rows and columns of matrices) and lower-case letters to refer to elements of vectors/matrices. For example, $\mathbf{X}$ is matrix, X_{i*} is the i-th row of $\mathbf{X}$, X_{*i} is the i-th column of $\mathbf{X}$ and x_{ij} is the j-th element of X_{i*}. The isolated vectors that do not belong to a matrix will be denoted with a capital letter, e.g., V.

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 according to their timestamps t_i. In the rest of the paper we omit the timestamps, since the values are ordered, and write the time series $X_1 = \{(0, 2), (1, 0), (2, -4)\}$ as the sequence $X_1 = [2, 0, -4]$. A *time series stream* S is an unbounded (i.e., never ending) time series. We write $\mathbf{X} = [X_{*1}|\ldots|X_{*m}]$ (or $\mathbf{X}_{n\times m}$) to denote an $n \times m$ matrix having m time series X_{*j} as columns and n values for each time series as rows.

A *sign vector* $Z \in \{1, -1\}^n$ is a sequence $[z_1, \ldots, z_n]$ of n unary elements, i.e., $|z_i| = 1$ for $i = \{1, \ldots, n\}$. We use $\times$ for scalar multiplications and $\cdot$ for matrix multiplications. The symbol $\|\|$ refers to the l-2 norm of a vector. Assuming $X = [x_1, \ldots, x_n]$, then $\|X\| = \sqrt{\sum_{i=1}^{n}(x_i)^2}$.

3.2 Background

As we mentioned earlier, ORBITS implements an incremental version of the Centroid Decomposition (CD) technique (cf. Figure 1). In what follows, we first describe CD and its most challenging part, the maximizing sign vector. Then, we introduce our problem definition.

3.2.1 Definition. CD is a matrix decomposition technique that decomposes an input $n \times m$ matrix $\mathbf{X}$ into an $n \times m$ loading matrix $\mathbf{L}$ and an $m \times m$ relevance matrix $\mathbf{R}$, such that $\mathbf{X} = \mathbf{L} \cdot \mathbf{R}^T$. CD is computed iteratively where at each iteration i ($i \in [1, m]$), a new maximizing sign vector Z_i (described below) is computed and used to compute the i-th relevance and loading vectors, i.e., $R_{*i} = \frac{\mathbf{X}^T \cdot Z_i}{\|\mathbf{X}^T \cdot Z_i\|}$ and $L_{*i} = \mathbf{X} \cdot R_{*i}$. Next, a matrix reduction step is applied in order to obtain the new relevance and loading vectors, i.e., $\mathbf{X} = \mathbf{X} - L_{*i} \cdot R_{*i}^T$. The decomposition returns m loading and relevance vectors, of size n and m respectively. The most challenging part of the CD of a matrix $\mathbf{X}$ is the computation of the vector Z.

3.2.2 Maximizing Sign Vector. Given an $n \times m$ matrix $\mathbf{X}$, the *maximizing* sign vector Z is the sign vector that maximizes the *centroid value* $\|\mathbf{X}^T \cdot Z\|$, i.e., Z satisfies the following equation:

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

The SSV algorithm [20] is the most efficient batch solution to solve Eq. (1). SSV is based on the derivation of a new and equivalent optimization problem:

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

where $diag^{=0}()$ is an auxiliary function that sets the diagonal values of a square matrix to 0, and $\mathbf{X} \cdot \mathbf{X}^T$ is the correlation matrix in case $\mathbf{X}$ is z-score normalized.

To solve the new optimization problem, the SSV algorithm starts by initializing a column vector $Z = [1, \ldots, 1]^T$ of length n, and iteratively computes a weight vector $V = diag^{=0}(\mathbf{X} \cdot \mathbf{X}^T) \cdot Z$. The elements of V are defined as:

$$v_i = z_i(z_i \times X_{i*} \cdot S - (X_{i*} \cdot (X_{i*})^T)) \tag{2}$$

where v_i is the i-th element of V and $S = \sum_{i=1}^{n}(z_i \times (X_{i*})^T)$.

Next, the signs of Z and V are compared, for each of their values $i \in \{0, \ldots, n\}$. SSV iteratively flips the sign of z_{pos} (the element at position pos in Z) from plus to minus, such that $|v_{pos} \times z_{pos}|$ is maximized. Once Z and V have the same pairwise sign in all positions, SSV terminates and returns the correct maximizing sign vector. Note that SSV flips the sign of only one element at a time.

3.3 Problem Definition

Let $\mathbf{X}$ be an $n \times m$ matrix of m time series each of length n and let $\Delta\mathbf{X}$ be an $r \times m$ increment to $\mathbf{X}$ with $r \ll n$. Let CD($\mathbf{X}$) be the Centroid Decomposition of $\mathbf{X}$. Our goal is to infer CD($\mathbf{X} + \Delta\mathbf{X}$) directly from CD($\mathbf{X}$) without recomputing it from scratch. Unlike incremental linear algebra frameworks such as Linview [29], which compute the change introduced by $\Delta\mathbf{X}$, our goal in this paper is to efficiently compute the whole decomposition of $\mathbf{X} + \Delta\mathbf{X}$.

4 CD OPTIMIZATION

Before introducing our incremental decomposition algorithm, we present a new Anticipatory Sign Vector (ASV) technique to speed up the sign vector search and, subsequently, the CD computation. Our approach for optimizing the sign vector search consists of reducing the number of weight vectors and thus reducing the number of iterations to compute the sign vectors. ASV is based on an incremental computation of weight vectors (Section 4.1) and makes use of an anticipatory termination (Section 4.2).

4.1 Incremental Weight Vector

The optimization of sign vectors computation resorts to efficiently finding weights vectors (cf. Eq. (2)). We propose an incremental technique to compute weight vectors. This technique makes it possible to anticipate the termination of the maximizing sign vector search, which considerably reduces the number of iterations.

Definition 1 (Weight vectors are incremental). *Let $Z^{(k)}$ be Z at iteration k, P the set of positions flipped in $Z^{(k)}$ and let v_i be the i-th weight value in V. For any two consecutive iterations of sign vectors, the weight vectors are linearly dependent, i.e.,*

$$v_i^{(k+1)} = v_i^{(k)} - 2 \times \sum_{p \in P \setminus \{i\}} (X_{i*} \cdot X_{p*}^T) \tag{3}$$

In the case where only one sign is flipped, $\forall i \in [1, n] \setminus \{p\}$, Eq.(3) can be rewritten as follows:

$$v_i^{(k+1)} = v_i^{(k)} - 2 \times (X_{i*} \cdot X_{p*}^T) \tag{4}$$

with $v_p^{(k+1)} = v_p^{(k)}$.

The incremental computation of weight vectors allows us to include a sequence of termination checks. Consequently, many weight vector computations do not have to be fully processed but can be interrupted once a termination condition is met. This incremental definition makes it possible to compute the correct maximizing sign vectors but in fewer iterations (compared to the SSV technique [20]). Note that in order to incrementally compute the weight vectors, the first weight vector is computed using Eq. (2).

Example 1. *To illustrate the incremental computation of the weight vectors, consider an input matrix $\mathbf{X}$ that contains two time series of five elements each, i.e.,*

$$\mathbf{X} = \begin{bmatrix} 5 & 1 \\ -10 & 5 \\ -9 & 4 \\ 4 & 6 \\ 2 & -4 \end{bmatrix}.$$

For the sake of simplicity, we illustrate the case where only one sign flip is performed. First, Z is initialized with 1s, i.e., $Z^{(1)} = \{1, 1, 1, 1, 1\}^T$ and the initial weight vector is computed using Eq.(2) to get $V^{(1)} = \{-54, 15, 23, -12, -84\}^T$. Three elements of $Z^{(1)}$ have a different sign from the corresponding elements in $V^{(1)}$ and among them the element in the 5th position has the highest absolute value. Using $p = 5$, the next weight vector is incrementally computed (using Eq. (4)) as follows

$$v_1 = -54 - 2 \times (\begin{bmatrix} 5 & 1 \end{bmatrix} \times \begin{bmatrix} 2 \\ -4 \end{bmatrix}) = -66$$

$$\vdots$$

$$v_5 = -84$$

i.e.,

$$Z^{(2)} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ -1 \end{bmatrix} \quad \text{and} \quad V^{(2)} = \begin{bmatrix} -66 \\ 95 \\ 91 \\ 20 \\ -84 \end{bmatrix}.$$

4.2 Anticipatory Sign Vector

Anticipatory sign vector is a new algorithm to efficiently compute the maximizing sign vector by including a sequence of termination checks. After every weight vector has been found, an anticipatory sign vector is computed, using the signs of the weight vector, and the flipping process is evaluated. This process can therefore be seen as a series of sign multi-flips where the final step returns the maximizing sign vector.

Definition 2 (Anticipatory sign vector). *Let $V^{(k)}$ be the weight vector obtained at iteration k. The vector $Z_A^{(k)}$ where each $z_i \in Z_A^{(k)}$ gets the same sign as $v_i \in V^{(k)}$ is called anticipatory sign vector.*

Lemma 1 (Anticipatory termination). *Let $Z_A^{(k)}$ be the anticipatory sign vector at iteration k and let $V_A^{(k)}$ be its corresponding weight vector. We can safely terminate the flipping process if the pairwise elements in $Z_A^{(k)}$ and $V_A^{(k)}$ have the same sign.*

Proof. We prove that the anticipatory termination yields the correct maximizing sign vector. This proof follows from the definition of the anticipatory sign vector. Since $Z_A^{(k)}$ contains all possible sign flips, then $(Z_A^{(k)})^T \cdot V_A^{(k)} \geq (Z^{(k+1)})^T \cdot V^{(k+1)}$. In case $Z_A^{(k)} = Z_A^{(k+1)}$ (termination condition), then $\nexists \; p$ s.t. $(Z_A^{(k+p)})^T \cdot V_A^{(k+p)} > (Z^{(k)})^T \cdot V^{(k)}$. Subsequently, $\forall p \in \mathbb{N}$, $(Z_A^{(k)})^T \cdot V_A^{(k)} = \max((Z^{(1)})^T \cdot V^{(1)}, \ldots, (Z^{(k)})^T \cdot V^{(k)}, \ldots, (Z^{(k+p)})^T \cdot V^{(k+p)})$. Thus, $Z_A^{(k)}$ is the maximizing sign vector. □

The anticipatory sign vector is computed as outlined in Algorithm 1. Given a matrix of m time series and an initial sign vector Z, we start by computing the initial weight vector V. We assume for now that Z contains only 1s and we show in Section 5.1 how to optimally initialize it. Next, the anticipatory sign vector is computed (line 3) and the weight vector V_A is initialized with the weight vector V (line 4). Then, we search for the set of positions of -1s in Z_A, called P, that have not been flipped yet (line 5). P is used to determine the positions that need to be flipped in the weight vector. And finally, V_A is updated according to Eq (3) (lines 7-8). If no position is left to be flipped in Z_A, i.e., $pos = 0$, then the anticipatory sign vector is returned as the maximizing one. Otherwise, the next weight vector is incrementally computed (lines 14-16). By using the anticipatory technique, we are able to find the required sign flips without going through the full sign flipping process.

Let us illustrate the anticipatory termination in the example depicted in Figure 2. The upper part of the figure shows the standard flipping process, SSV, while the lower part shows the anticipatory termination, ASV. We consider the same input matrix $\mathbf{X}$ of our running example. As illustrated in the upper part of the figure, we start computing the weight values for each weight vector using Eq. (2). The last element in the first weight vector $V^{(1)}$ has the smallest negative (and sign opposite) value and thus, the sign of the element in position 5 in Z is flipped. The standard approach applies the same procedure until no more elements need to be flipped (i.e., $Z^{(4)}$). Using ASV on the other hand, as illustrated in the lower part of the figure, $Z_A^{(1)}$ gets the same sign as the elements in $V^{(1)}$ and subsequently $V_A^{(1)}$ is computed. The termination condition is met, i.e., $Z_A^{(1)}$ and $V_A^{(1)}$ have the same pairwise element sign. Hence, ASV may immediately stop the flipping process, whereas the standard flipping process takes three additional iterations to terminate. As illustrated in this example, ASV returns the same maximizing sign vector while requiring less than half of the number of iterations.

Algorithm 1: ASV($\mathbf{X}, Z$)

Input : $n \times m$ matrix $\mathbf{X}$, sign vector Z
Output: maximizing sign vector $Z \in [1, -1]^n$

1 $V \coloneqq$ Compute initial weight vector ; ▷ using Eq.(2)
2 **repeat**
3 $\quad Z_A \coloneqq \{z_i^* \in Z_A \mid z_i^* = \frac{v_i}{|v_i|}\}$;
4 $\quad V_A \coloneqq V$;
5 $\quad P \coloneqq \{i \mid z_i \times z_i^* = -1 \,\&\, z_i \in Z \,\&\, z_i^* \in Z_A\}$;
6 $\quad$ **foreach** $k \in P$ **do**
7 $\quad\quad V_A \coloneqq V_A - 2 \times \mathbf{X} \cdot X_{k*}$; ▷ using Eq.(3)
8 $\quad\quad v_k^* \coloneqq v_k^* + 2 \times X_{k*} \cdot X_{k*}$
$\quad\quad$ // v_k^* is the k-th element of V_A
9 $\quad pos \coloneqq \{i \mid v_i^* = \max|(v_j^* \in V_A)| \,\&\, z_j^* \times v_j^* < 0\}$;
$\quad$ ▷ Anticipatory termination
10 $\quad$ **if** $pos = 0$ **then**
11 $\quad\quad p \coloneqq pos$;
12 $\quad\quad Z \coloneqq Z_A$;
13 $\quad$ **else**
14 $\quad\quad p \coloneqq \{i \mid v_i = \max|(v_j \in V)| \,\&\, z_j \times v_j < 0\}$;
15 $\quad\quad z_p \coloneqq (-1) \times z_p$;
16 $\quad\quad V \coloneqq$ Compute incremental weight vector;
17 **until** $p = 0$;
18 **return** $Z \in [1, -1]^n$;

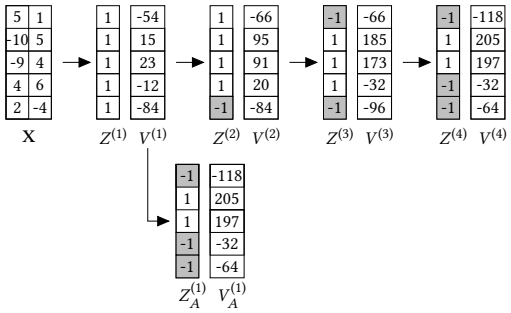


Figure 2: Anticipatory Termination Example.

5 INCREMENTAL CENTROID DECOMPOSITION

Our incremental Centroid Decomposition is a two-fold procedure. First, it applies an incremental initialization of sign vectors, which leverages the matrix similarity before and after each time series update. Second, it uses the ASV algorithm to efficiently find the corresponding sign vectors. We begin by introducing the initialization technique, before introducing our incremental algorithm and proving its correctness. Finally, we present ORBITS, which uses the incremental CD technique to recover blocks of missing values.

5.1 Sign Vector Initialization

The intuition behind our incremental initialization revolves around the idea that matrices before and after each row update share similar properties. This similarity reflects the temporal continuity in time series. Our proposed approach leverages this *matrix similarity* to

compute the maximizing sign vectors of the updated matrix using the sign vectors cached from the previous CD computation.

Lemma 2 (Matrix Similarity). *Let $\widetilde{\mathbf{X}}$ be the matrix resulting from incrementing an $n \times m$ matrix $\mathbf{X}$ with an $r \times m$ matrix $\Delta\mathbf{X}$ and let A_{i*} be the i-th row of $\Delta\mathbf{X}$. Let also $Z \in \{-1, 1\}^n$ and $\tilde{Z} \in \{-1, 1\}^{n+r}$ be two sign vectors. Then, the following holds:*

$$\max\|\widetilde{\mathbf{X}}^T \cdot \tilde{Z}\| = \max\|\mathbf{X}^T \cdot Z\| + \sum_{i=1}^{r} \|A_{i*}\|$$

Proof. The proof is given in Appendix A. □

Lemma 2 shows the relationship between the maximizing sign vector before any row update of $\mathbf{X}$ and the one after the update. It states that each update of the input matrix augments the centroid values with the norm of the added data. We use the sign vector computed before the update as input to find the maximizing sign vector of the incremented matrix $\widetilde{\mathbf{X}}$. This initialization takes into consideration the fact that the length of the initial sign vector might not match the dimensions of $\widetilde{\mathbf{X}}$; the input matrix might have grown since the last computation of the algorithm and might thus require larger sign vectors than those already computed. We append additional 1s (since the sign vectors consist of 1s and -1s only) at the end of the sign vector until reaching the correct length. Note that appending -1s yields the opposite sign vector.

We now use the matrix similarity property to prove that our technique computes the correct sign vector, which guarantees an accurate recovery.

Theorem 1 (Correctness). *Let $\widetilde{\mathbf{X}}$ be the resulting matrix of incrementing $\mathbf{X}$ with $\Delta\mathbf{X}$. Then, our technique returns the sign vector, $\tilde{Z}$, that maximizes $\|\widetilde{\mathbf{X}}^T \cdot \tilde{Z}\|$.*

Proof. Using $Z^{(1)}$ as Z at the first iteration of the algorithm, we introduce the two following vectors. Let $\tilde{Z}$ be the resulting sign vector obtained by batch CD (i.e., $\tilde{Z}^{(1)} = [1, \ldots, 1]$) and let $\tilde{Z}_c$ be the resulting sign vector obtained by our technique with (i.e., $\tilde{Z}_c^{(1)} = [Z_c, 1, \ldots, 1]$; Z_c is the cached sign vector). Proving the correctness of our technique boils down to proving that:

$$\underset{\tilde{Z}_c \in \{-1,1\}^{(n+r)}}{arg\ max} \|\widetilde{\mathbf{X}}^T \cdot \tilde{Z}_c\| \equiv \underset{\tilde{Z} \in \{-1,1\}^{(n+r)}}{arg\ max} \|\widetilde{\mathbf{X}}^T \cdot \tilde{Z}\|$$

Let $\mathbf{I}$ be an identity matrix, $\mathbf{D}$ be a diagonal matrix containing $\tilde{Z}_c^{(1)}$, i.e., $\mathbf{D} = diag(\tilde{Z}_c^{(1)})$, and let $\widetilde{\mathbf{X}}_D$ be an $(n + r) \times m$ matrix s.t. $\widetilde{\mathbf{X}}_D^T = \widetilde{\mathbf{X}}^T \cdot \mathbf{D}$. Let also $\tilde{Z}_D$ be an $(n + r)$ sign vector s.t $\tilde{Z}_D = \mathbf{D} \cdot \tilde{Z}_c$.

First, we prove the following:

$$\underset{\tilde{Z}_c \in \{-1,1\}^{(n+r)}}{arg\ max} \|\widetilde{\mathbf{X}}^T \cdot \tilde{Z}_c\| \equiv \underset{\tilde{Z}_D \in \{-1,1\}^{(n+r)}}{arg\ max} \|\widetilde{\mathbf{X}}_D^T \cdot \tilde{Z}_D\|$$

Since $\mathbf{D}$ is a signature matrix where the diagonal elements are +1 or –1, then $\mathbf{D} \cdot \mathbf{D} = \mathbf{I}$. It follows:

$$\underset{\tilde{Z}_c \in \{-1,1\}^{(n+r)}}{arg\ max} \|\widetilde{\mathbf{X}}^T \cdot \tilde{Z}_c\| \equiv \underset{\tilde{Z}_c \in \{-1,1\}^{(n+r)}}{arg\ max} \|\widetilde{\mathbf{X}}^T \cdot \mathbf{D} \cdot \mathbf{D} \cdot \tilde{Z}_c\| \quad (5)$$

By definition of $\mathbf{D}$, we have $d_{ii} \times z_i^{(1)} = 1, \forall i \in \{1, \cdots, (n + r)\}$ where $z_i^{(1)} \in \tilde{Z}_c^{(1)}$, which yields $\mathbf{D} \cdot \tilde{Z}_c^{(1)} = [1, \ldots, 1]$. Since $\tilde{Z}_D^{(1)} = \mathbf{D} \cdot \tilde{Z}_c^{(1)}$, we replace the argument $\tilde{Z}_c$ by $\tilde{Z}_D$ and get

$$\underset{\tilde{Z}_c \in \{-1,1\}^{(n+r)}}{arg\ max} \|\widetilde{\mathbf{X}}^T \cdot \mathbf{D} \cdot \mathbf{D} \cdot \tilde{Z}_c\| \equiv \underset{\tilde{Z}_D \in \{-1,1\}^{(n+r)}}{arg\ max} \|(\widetilde{\mathbf{X}}^T \cdot \mathbf{D}) \cdot \tilde{Z}_D\|$$
$$\equiv \underset{\tilde{Z}_D \in \{-1,1\}^{(n+r)}}{arg\ max} \|\widetilde{\mathbf{X}}_D^T \cdot \tilde{Z}_D\| \quad (6)$$

Putting (5) into (6), we get

$$\underset{\tilde{Z}_c \in \{-1,1\}^{(n+r)}}{arg\ max} \|\widetilde{\mathbf{X}}^T \cdot \tilde{Z}_c\| \equiv \underset{\tilde{Z}_D \in \{-1,1\}^{(n+r)}}{arg\ max} \|\widetilde{\mathbf{X}}_D^T \cdot \tilde{Z}_D\| \quad (7)$$

Next, we prove the following

$$\underset{\tilde{Z} \in \{-1,1\}^{(n+r)}}{arg\ max} \|\widetilde{\mathbf{X}}^T \cdot \tilde{Z}\| \equiv \underset{\tilde{Z}_D \in \{-1,1\}^{(n+r)}}{arg\ max} \|\widetilde{\mathbf{X}}_D^T \cdot \tilde{Z}_D\|$$

By definition of $\widetilde{\mathbf{X}}_D$ we have

$$\widetilde{\mathbf{X}}_D^T = \widetilde{\mathbf{X}}^T \cdot \mathbf{D}$$
$$= \begin{bmatrix} \mathbf{X} \\ \Delta\mathbf{X} \end{bmatrix}^T \cdot \begin{bmatrix} diag(Z_c) & \mathbf{0}_{n \times r} \\ \mathbf{0}_{r \times n} & \mathbf{I}_{r \times r} \end{bmatrix}$$
$$= \begin{bmatrix} (\mathbf{X}^T \cdot diag(Z_c)) & \Delta\mathbf{X}^T \end{bmatrix} \quad (8)$$

From (8), we can see that $\widetilde{\mathbf{X}}_D$ is $diag(Z_c)^T \cdot \mathbf{X}$ incremented with $\Delta\mathbf{X}$. By applying Lemma 2 on $\widetilde{\mathbf{X}}_D$ we get

$$\max\|\widetilde{\mathbf{X}}_D^T \cdot \tilde{Z}_D\| = \max\|\mathbf{X}^T \cdot diag(Z_c) \cdot Z\| + \sum_{i=1}^{r} \|A_{i*}\|$$
$$= \max\|\mathbf{X}^T \cdot Z\| + \sum_{i=1}^{r} \|A_{i*}\| \quad (9)$$

Using (9) and Lemma 2, we get

$$\max\|\widetilde{\mathbf{X}}^T \cdot \tilde{Z}\| = \max\|\widetilde{\mathbf{X}}_D^T \cdot \tilde{Z}_D\| \quad (10)$$

Since the two equations in (10) use the same initial maximizing sign vector that contains only 1s. Thus, (10) can be rewritten as follows:

$$\underset{\tilde{Z} \in \{-1,1\}^n}{arg\ max} \|\widetilde{\mathbf{X}}^T \cdot \tilde{Z}\| = \underset{\tilde{Z}_D \in \{-1,1\}^n}{arg\ max} \|\widetilde{\mathbf{X}}_D^T \cdot \tilde{Z}_D\| \quad (11)$$

By transitivity of (7) and (11), we get

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

Therefore, our technique computes the same maximizing sign vector as the batch CD, which concludes the proof. □

Example 2. *To illustrate the incremental initialization after appending multiple rows, consider the input matrix of our running example. The two time series have been sequentially appended with (0, 8) and (2,7), respectively. Figure 3 illustrates the initialization and the computation of the maximizing sign vector. For the sake of simplicity, we illustrate the flipping at the same time of one element only. To compute the first maximizing sign vector of $\widetilde{\mathbf{X}}$, we use the sign vectors computed before the update. The incremental initialization first takes the (cached) sign vector Z_1 as the initial sign vector. Then, it appends two 1s at the end of each cached sign vector and starts flipping the signs to find the maximizing sign vectors for $\widetilde{\mathbf{X}}$. At each iteration k and for each sign vector, a pair of vectors $\tilde{Z}^{(k)}$ and $\tilde{V}^{(k)}$ is computed. The weight vectors V are incrementally computed according to Eq* (3) *(similarly to Example 1).*

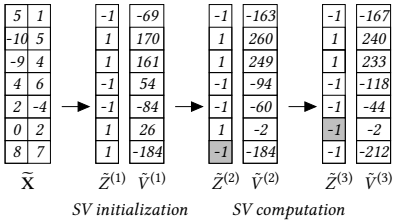


Figure 3: Incremental Sign Vectors.

Among all elements in $\tilde{V}^{(1)}$ which have a different sign from their corresponding ones in $\tilde{Z}^{(1)}$, the element in position 7 in $\tilde{V}^{(1)}$ has the highest absolute value. Thus, in the first iteration, we flip the sign of the element at position 7 in $\tilde{Z}^{(1)}$ and we use the new sign vector $\tilde{Z}^{(2)}$ to compute $\tilde{V}^{(2)}$. Next, we find that the element at position 6 of $\tilde{Z}^{(2)}$ is the only element that has a different sign from its corresponding element in $\tilde{V}^{(2)}$. Thus, in the second iteration, we flip the sign of the element at position 6 in $\tilde{Z}^{(3)}$ and we use the new sign vector $\tilde{Z}^{(3)}$ to compute $\tilde{V}^{(3)}$. Since all corresponding elements in $\tilde{Z}^{(3)}$ and $\tilde{V}^{(3)}$ have the same pairwise sign, the algorithm terminates and returns $\tilde{Z}^{(3)}$ as the first maximizing sign vector. We apply the same procedure to compute the second maximizing sign vector.

5.2 Putting It All Together

We now present the InCD algorithm that implements our incremental solution. First, the sign vector used to initialize the search for the maximizing sign vector is read from cache (line 2) and its size is adjusted. Then, matrix $\mathbf{X}$ (which is already loaded in memory) is appended with the stream of rows $\Delta\mathbf{X}$ to form $\widetilde{\mathbf{X}}$. The latter is used to initialize the weight vector (line 6). Next, we use our anticipatory algorithm to compute the maximizing sign vector of $\widetilde{\mathbf{X}}$, which is then written back to the cache (line 7).

Algorithm 2: InCD($\Delta\mathbf{X}, r, m$)

Input : $r \times m$ matrix of appended rows $\Delta\mathbf{X}$
Output: $n \times m$ matrix $\mathbf{L}$, $m \times m$ matrix $\mathbf{R}$

1 **for** $i = 1$ **to** m **do**
  `// incremental initialization`
2 $Z_i \coloneqq$ read sign vector of $\mathbf{X}$ from cache position i;
3 **while** $length(Z_i) < n$ **do**
4 append 1 as row value to Z_i;
5 $\widetilde{\mathbf{X}} \coloneqq$ increment $\mathbf{X}$ with $\Delta\mathbf{X}$;
  `// Anticipatory search`
6 $\tilde{Z}_i \coloneqq ASV(\widetilde{\mathbf{X}}, Z_i)$;
7 $cache(\tilde{Z}_i, i)$;
  `// calculation of the decomposition`
8 $R_{*i} \coloneqq \frac{\widetilde{\mathbf{X}}^T \cdot \tilde{Z}}{\|\widetilde{\mathbf{X}}^T \cdot \tilde{Z}\|}$; $L_{*i} \coloneqq \widetilde{\mathbf{X}} \cdot R_{*i}$;
9 $\widetilde{\mathbf{X}} \leftarrow \widetilde{\mathbf{X}} - L_{*i} \cdot R_{*i}^T$;
10 **return** $\mathbf{L}, \mathbf{R}$;

InCD reads and writes in-place to the same cache position yielding a minimal cache footprint. Assuming that the input matrix $\mathbf{X}$ consists of float-precision (4 byte) values, the cache size required to store the maximizing sign vectors is only $^1/_{16}$ of the memory size required to store $\mathbf{X}$, preventing any cache overflow. Finally, the found maximizing sign vector is used to sequentially compute the decomposition of $\widetilde{\mathbf{X}}$.

The runtime complexity of InCD is dominated by the call of `ASV()`, which is performed m times. ASV uses the same definition of weight vectors as the SSV algorithm and thus, an element that has been flipped can not be flipped again (see Lemma 5 in [20]). Subsequently, ASV requires $2n$ iterations to compute the weight vectors (Lines 7-8 and Line 16 in ASV) and p iterations to switch the sign values (Lines 14-16 in ASV), yielding $O(pn)$ time. Since we know from Lemma 2 that the change in the centroid values (and consequently the sign vectors) is bounded by the norm of the added rows (r), then the time complexity to compute p is $O(r)$. Also, as we are dealing with long (streams) of time series, then $n \gg m$ and $n \gg r$. Hence, to decompose $\widetilde{\mathbf{X}}$, InCD yields a time complexity of $O(n)$. At any time, there are exactly m sign vectors from the last computation of InCD stored in cache. Thus, our algorithm keeps in memory m sign vectors as binary arrays, as well as the updated matrix $\widetilde{\mathbf{X}}$, each with an $O(n)$ space complexity. Therefore, the total space complexity is also $O(n)$.

The following section introduces ORBITS, that relies on InCD to recover missing values in an online fashion.

5.3 Recovery of Missing Values (ORBITS)

Algorithm 3: ORBITS ($\mathbf{X}$, $\Delta\mathbf{X}$, T^-)

Input : $n \times m$ input matrix $\mathbf{X}$; $r \times m$ matrix of appended rows $\Delta\mathbf{X}$; List of missing time points T^-
Output: Matrix with recovered values $\widetilde{\mathbf{X}}'$

1 $\widetilde{\mathbf{X}} \coloneqq$ increment $\mathbf{X}$ with $\Delta\mathbf{X}$;
2 Linearly interpolate/extrapolate all missing values in $\widetilde{\mathbf{X}}$;
3 **repeat**
4 $\widetilde{\mathbf{X}}' \coloneqq \widetilde{\mathbf{X}}$;
5 compute reduction factor k of $\widetilde{\mathbf{X}}$;
6 $\mathbf{L}_k, \mathbf{R}_k \coloneqq InCD(\Delta\mathbf{X}, r, m-k)$;
7 $\widetilde{\mathbf{X}}_k \coloneqq \mathbf{L}_k \cdot (\mathbf{R}_k)^T$;
  `// Update missing values`
8 **foreach** $(i, j) \in T^-$ **do**
9 $\tilde{x}_{ij} \coloneqq y_{ij}$;
  `//` y_{ij} `element of` $\widetilde{\mathbf{X}}_k$ `at timestamp` i
10 **until** $\|\widetilde{\mathbf{X}} - \widetilde{\mathbf{X}}'\|_{\hat{F}} < \epsilon$;
11 **return** $\widetilde{\mathbf{X}}'$;

Algorithm 3 describes the pseudocode of ORBITS. It takes as input a matrix $\mathbf{X}$ incremented with $\Delta\mathbf{X}$ (both matrices might contain a set of missing blocks, B) and a list T^- of pairs indicating the rows and columns of the missing values in $\mathbf{X}$. The recovery starts by initializing the missing values in $\widetilde{\mathbf{X}}$ using either linear interpolation or zero, depending on the position of the missing values in $\widetilde{\mathbf{X}}$. Then, we apply a reduction to the decomposition of $\widetilde{\mathbf{X}}$ to return $\mathbf{L}_k$ and $\mathbf{R}_k$, which contain the first $m-k$ columns of $\mathbf{L}$ and $\mathbf{R}$ respectively. We utilize the commonly used entropy method to dynamically set, at each iteration, the reduction factor k. We use the same procedure that applies to SVD [2], but we consider the centroid values instead of the eigenvalues [21]. Next, the values in $\widetilde{\mathbf{X}}$ with positions in

T^- are updated with their corresponding ones in $\widetilde{\mathbf{X}}_k = \mathbf{L}_k \cdot \mathbf{R}_k^T$. The recovery process terminates if the difference in the relative Frobenius norm $\|\widetilde{\mathbf{X}} - \widetilde{\mathbf{X}}'\|_{\hat{F}} = \sqrt{\sum_{i=1}^{|B|} \sum_{j=1}^{m} (\tilde{x}_{ij} - \tilde{x}'_{ij})^2 / |B|}$ between $\widetilde{\mathbf{X}}'$ and $\widetilde{\mathbf{X}}$ (where $\tilde{x}_{ij} \in \widetilde{\mathbf{X}}$, $\tilde{x}'_{ij} \in \widetilde{\mathbf{X}}'$ and |B| is the length of the missing block) falls below a small threshold value ϵ. We empirically set the value of ϵ to 10^{-6}, which yields the best accuracy/precision trade-off. Since the entropy, used to find the reduction value k, minimizes the Frobenius norm as does the iterative process, then our recovery algorithm quickly converges (see Figure 4 in [21]).

6 EXPERIMENTAL EVALUATION

In this section, we report our experimental results. We aim to evaluate: (1) the performance of ORBITS against the state-of-the-art online recovery techniques and (2) the impact of our sign vector computation on ORBITS's performance.

6.1 Setup and Datasets

We rewrote all algorithms in C++, except for MRNN (inextricable from Python), using an advanced linear algebra library called Armadillo [33]. We re-engineered the original implementations, which led to gains in performance across all the algorithms – in one case making an algorithm (SPIRIT) 110x faster. We implemented all recovery algorithms in a non-distributed fashion as they rely on incremental computations, which are not easy to parallelize.

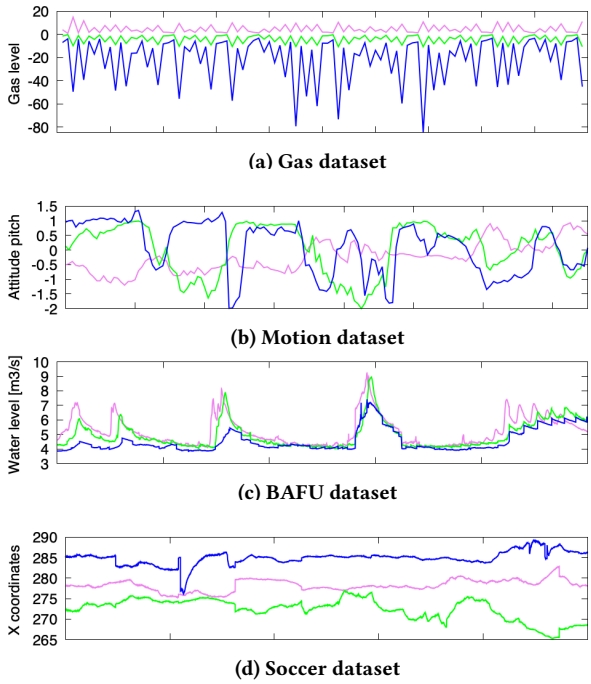
All the experiments were ran on a Linux machine with a 3.4 GHz Intel Core i7 and 128GB of RAM. All code and data (and additional experiments omitted for brevity) are publicly available[1]. Our evaluation was performed on real-world time series from a broad range of applications and which cover a wide range of characteristics and sizes (cf. Table 1). We used the following datasets:

- **Soccer.** This dataset was introduced in the DEBS challenge 2013 [28] and contains the position of players during a football match. The data originates from sensors located near the players' shoes and the goal keeper hands. This dataset provides values recorded at very a high rate, i.e., the tracking frequency is equal to 200Hz yielding 15'000 position events per second. Soccer time series are bursty with lots of outliers.
- **MotionSense.** This dataset includes time series data generated by accelerometer and gyroscope sensors (attitude, gravity, userAcceleration, and rotationRate) [24]. The sensors measure different human activities at a high sampling rate of 50Hz using an iPhone 6s kept in the users' front pockets. The motion time series are non-periodic, but exhibit partial trend similarities.
- **BAFU.** This dataset consists of water discharge time series provided by the BundesAmt Für Umwelt (BAFU) [1], the Swiss Federal Office for the Environment, collected from different Swiss rivers. The frequency of measurements is high to track the fast-changing water pressure during wet seasons. This dataset contains periodic time series where some of them are shifted in time.
- **Gas.** Gas concentration collected from a smart gas delivery platform facility situated at the ChemoSignals Laboratory at the University of California in San Diego [32]. This dataset is used to measure the impact of sustainable green infrastructures and contains different gases that present mixed correlation (positive/negative and high/low).

The BAFU and Gas datasets were used in a recent benchmark that evaluates missing values recovery techniques [22]. The values of the observations are stored as 4-byte floating numbers while the sign vectors are binary arrays. All the time series have been z-score normalized.

Table 1: Description of time series.

Name	TS length	# of TS	Main features
Gas	1000	100	mixed correlations
Motion	10'000	20	non-periodic, local similarity
BAFU	50'000	10	periodic, time-shifts
Soccer	500'000	10	sudden change, anomalies



(a) Gas dataset

(b) Motion dataset

(c) BAFU dataset

(d) Soccer dataset

Figure 4: Three sample time series from each dataset.

6.2 Recovery of Missing Values

In this section, we evaluate ORBITS under a variety of recovery scenarios. We compare its accuracy and efficiency against the state of the art in streaming recovery techniques, i.e., OGDImpute [3], SAGE [5], pcaMME [27], SPIRIT [31], TKCM [36] and MRNN [38]. For the parametric techniques, we use the parameters recommended by the authors. We report the results over three runs. To measure the accuracy, we adopt the most commonly used measure in this

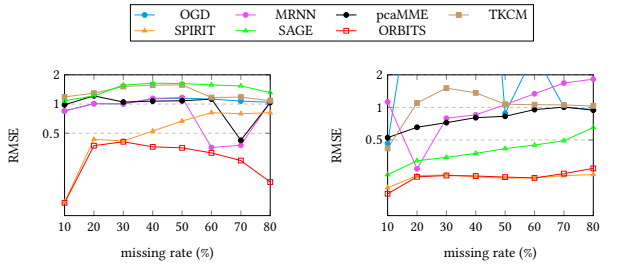
[1] https://github.com/eXascaleInfolab/orbits

field [25]: the root mean square error (RMSE) between the original blocks and the recovered ones, i.e.,

$$RMSE = \sqrt{\frac{1}{T}\sum_{t \in T}(x_t - \tilde{x}_t)^2}$$

where T is the set of missing values, x_t is the original value and $\tilde{x}_t$ is the recovered value.

6.2.1 Accuracy. We first consider the simplest case where a single time series contains only one missing block. To simulate an online recovery, we set the missing block to appear at the tip of a randomly chosen series and we vary the size of the missing block from 10% to 80% starting from the end of the series. We keep the length and number of the time series to their maximum per dataset. Figure 5 shows the RMSE results when varying the missing rate. We exclude MRNN and TKCM from the soccer experiment as both algorithms take more than 15 hours to complete one tick.



(a) Gas dataset.

(b) Motion dataset. ORBITS and SPIRIT are overlapping.

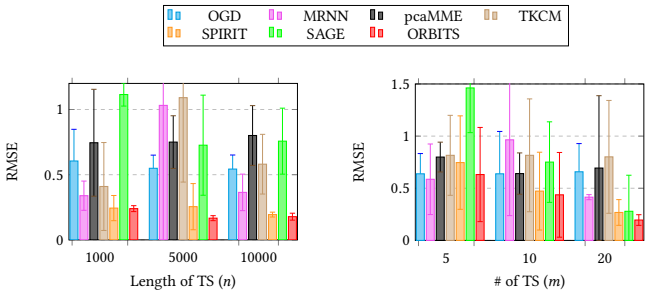
(c) BAFU dataset.

(d) Soccer dataset. MRNN and TKCM are excluded.

Figure 5: Accuracy comparison with increasing missing block size.

First, we observe that ORBITS outperforms in general all recovery baselines and yields 25% and 30% RMSE improvement compared to SPIRIT and SAGE, respectively. The only exception is on the Motion dataset, where our technique achieves a similar recovery as SPIRIT. This is explained by the fact that this dataset presents local similar trends to which both techniques respond in a similar fashion. The hybrid principal components computed by SPIRIT in this case carry similar information to the loading vectors computed by ORBITS. Second, in the datasets with a small number of series, i.e, Motion, BAFU and Soccer, as expected, the RMSE of ORBITS increases with the missing rate. Surprisingly, in the Gas dataset, the RMSE shows a bell shape as the missing rate grows. This is explained by the fact that in the presence of a high number of series, larger missing blocks require a higher number of iterations which, in turn, yield better recovered values.

Next, we evaluate the recovery accuracy on different datasets when increasing the dataset size. We assume one incomplete series with a missing block of size 10% of the maximum series' length. When the sequence length varies, the number of series is set to 10, while the sequence length is set to 1k values when the number of series varies. We use the average RMSE values across different datasets, with standard deviation as the confidence intervals. Figure 6 depicts the results.



(a) Avg. RMSE with varying length of TS.

(b) Avg. RMSE with varying # of TS.

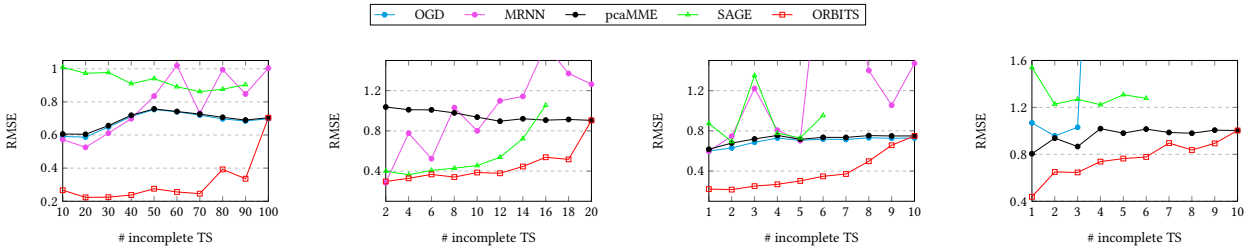
Figure 6: Accuracy comparison with increasing time series length and number.

We observe that most techniques take advantage of using longer series to achieve a lower error (cf. Figure 6a). The improvement is more noticeable when we vary the length either from 1k to 5k or from 5k to 10k, depending on the technique. This is expected, because using more data helps these techniques to better capture the main features of the data. We observe also that ORBITS achieves the lowest standard deviation as its RMSE is steady across datasets.

In the experiments of Figure 6b, we observe that all techniques take advantage of using more time series per dataset. The results show also that ORBITS outperforms all techniques in all datasets. The only exception is when the number of time series is equal to five where MRNN achieves a slightly lower RMSE than ORBITS (0.59 vs 0.63). This is explained by the fact that, for such a low number of time series, MRNN produces a recovery that mimics the average, which turns out to yield a low error. However, compared to ORBITS, the recovery of MRNN does not preserve the shape of the missing block (as we will show later in this section).

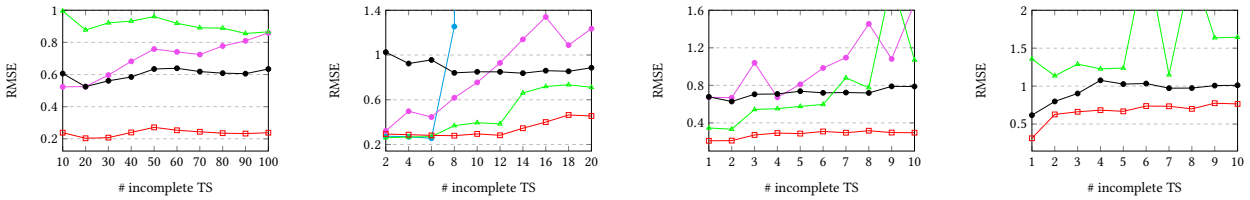
In the previous set of experiments, we assumed that a missing block occurs in a single series in a dataset. In real-world applications, however, more than one sensor might break during multiple time periods. This yields multiple incomplete time series with one or more missing blocks each.

In the experiments in Figure 7, we evaluate the case of multiple incomplete time series where each has one missing block at the end (of size of 10% of the series). We keep the length and number of series to their maximum per dataset and we vary the number of incomplete series. TKCM and SPIRIT cannot handle more than one incomplete series and are hence discarded (cf. Section 2). SAGE requires every row of the input matrix to contain at least k non-missing values, where k is the reduction factor. Thus, this technique can process only a certain number of incomplete



(a) Gas dataset. OGD and pcaMMe are overlapping. **(b) Motion dataset. OGD is off the scale.** **(c) BAFU dataset.** **(d) Soccer dataset.**

Figure 7: Accuracy with increasing number of incomplete series (single missing block case).



(a) Gas dataset. OGD is off the scale **(b) Motion dataset.** **(c) BAFU dataset. OGD is off the scale.** **(d) Soccer dataset. OGD is off the scale.**

Figure 8: Accuracy with increasing number of incomplete series (MCAR case).

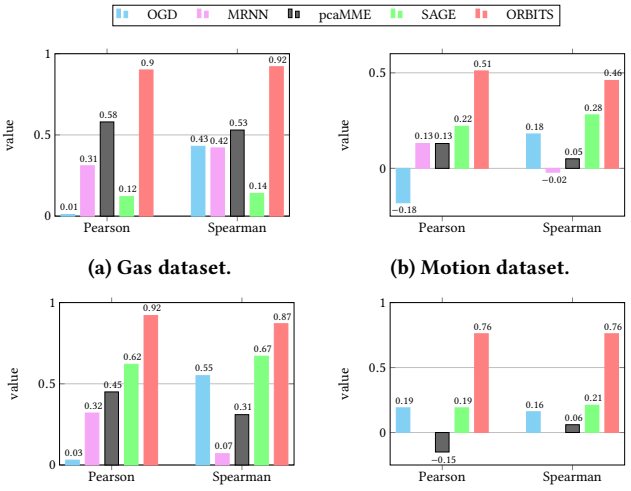
series in the dataset. We exclude MRNN from the soccer experiment for the same reason explained before, completion time.

The results show that our technique achieves the lowest RMSE when varying the number of incomplete series. As expected, the RMSE of ORBITS increases with the number of incomplete series. This is explained by the fact that more incomplete series means more values to recover, which in turn yields a higher error. We note that the last point in the plot corresponds to a blackout, i.e, when all time series lose data at the same time. In this case, ORBITS terminates in few iterations, incurring a high RMSE. The recovery of ORBITS in the case of blackout is the same as pcaMMe because both techniques use similar initialization of the missing blocks.

Next, we consider the most frequent scenario where each incomplete time series has multiple missing blocks (of the total size of 10% per series). We set the missing blocks to appear completely at random (MCAR), i.e., missing blocks can be overlapping, disjoint or both at the same time. We keep the length and number of series to their maximum per dataset and vary the number of incomplete series. The results are depicted in Figure 8.

We observe that ORBITS achieves the lowest RMSE in all datasets and is on average between 30% (cf. Figure 8d) and 60% (cf. Figure 8a) more accurate than its closest competitor. Surprisingly, increasing the number of series did not always increase the RMSE of ORBITS. The reason is that the MCAR scenario yields relatively small blocks, which poses no challenge to our technique.

In addition to the RMSE analysis, we also compare the accuracy of the evaluated techniques to reconstruct the shape of the missing blocks. We compute the correlation between the original blocks and the recovered ones using Pearson and Spearman coefficients [11]. The two metrics capture different trends. The former captures linear relationships (e.g, aligned trends) while the latter captures non-linear relationships (e.g, shifted trends). The two metrics range between 1 and -1, where 1 stands for perfect positive correlation, -1 for perfect opposite correlation, and 0 for no correlation.



(a) Gas dataset. **(b) Motion dataset.** **(c) BAFU dataset.** **(d) Soccer dataset.**

Figure 9: Shape reconstruction.

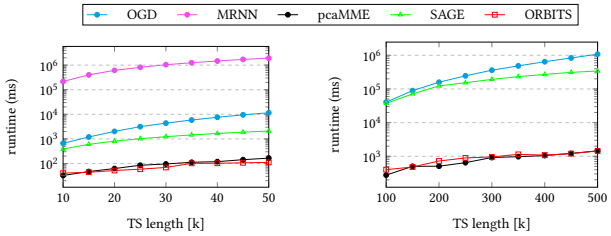
Figure 9 depicts the results of the correlation experiment where each time series has missing MCAR multiblocks of a total size of 10% per series. The results show that ORBITS achieves the highest Pearson and Spearman correlation values, preserving the shape of the original block. This is attributed to the weight vectors, computed

during the decomposition, which embed the correlation across the input series. We can also observe that only ORBITS and SAGE have on par Pearson and Spearman results: time shifts do not affect the accuracy of the recovery of both techniques. The remaining techniques are sensitive to non-linear relationships (e.g., OGD and MRNN have a close to zero Spearman correlation on the BAFU dataset). All techniques achieve their lowest correlation on the motion dataset, because motion time series contain a high number of irregular fluctuations.

6.2.2 Efficiency. We now evaluate the efficiency of the recovery techniques by measuring the runtime while varying the sequence length and number. We report the time on a log scale, since the results widely vary among algorithms.

In Figure 10 we incrementally increase the stream length, using all the series from the dataset. We consider the MCAR scenario on the BAFU and the Soccer datasets, which have the longest time series, 50k and 500k respectively.

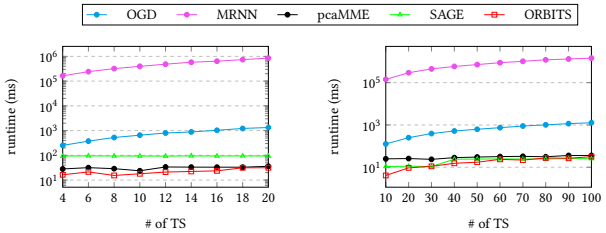
The results show that ORBITS and pcaMME are the two most efficient techniques and are orders of magnitude faster than the remaining techniques. pcaMME relies on an approximation of the principal components yielding a low runtime. However, this approximation incurs a high recovery error (as shown earlier). Unlike pcaMME, our technique relies on a fast incremental computation of the centroid values rendering our recovery very efficient. The incremental computation computes the correct decomposition, which explains the low recovery error. ORBITS achieves a linear time complexity and takes 1.4 sec to recover 10 time series each with 500k (cf. Figure 10b).



(a) BAFU dataset (n = 50k). **(b) Soccer dataset(n =500K).**

Figure 10: Efficiency with increasing series length.

In the experiments of Figure 11, we gradually increase the number of time series in a dataset while keeping their length fixed at their maximum. We choose the Gas and the Motion datasets for the same reason as above (their size). We observe the same trends as in Figure 10. ORBITS stands out among the faster peers when using a few series per dataset. Our technique achieves a sub-linear complexity with the number of series. The efficiency of our technique is explained by the careful computation of the appropriate reduction factor. At each iteration, ORBITS finds the reduction factor that yields the best accuracy/precision trade-off. For larger number of series (e.g., 100), the runtime difference between our technique and the PCA-based ones becomes indistinguishable. For a large number of series, the centroid vectors (ORBITS) and the principal components (pcaMMe and SAGE) require a similar number of iterations to be computed and, subsequently, the same runtime.

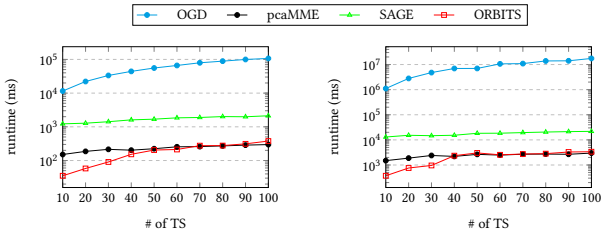


(a) Motion dataset (m = 20). **(b) Gas dataset (m = 100).**

Figure 11: Efficiency with increasing series number.

Next, we evaluate the impact of series' number on the recovery runtime using large-scale datasets. We exclude MRNN from this experiment for the same reason mentioned earlier. We augment both the BAFU and Soccer datasets with more time series. To do so, we first generate synthetic time series that have similar properties as the existing ones using the Weighted_DBA technique [12]. Next, we alter the time series with noise following a Gaussian distribution $N(0, 0.2)$ and with exponential smoothing. This allows us to obtain time series that exhibit similar shape but with some local dissimilarities. Finally, we apply a full permutation of blocks of ten series. The results are depicted in Figure 12.

We observe a similar trend as in Figure 11, ORBITS stands apart among the rest of the techniques when using a few series (cf. Figures 12a and 12b). The runtime difference between ORBITS and pcaMME becomes, however, indistinguishable as the number of series increases. SAGE becomes slower because SGD is more expensive to compute on larger matrices. It is worth to notice that the efficiency of pcaMME comes with the cost of a low accuracy.



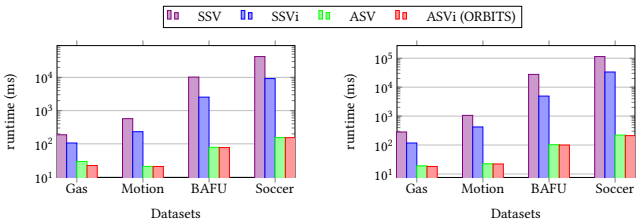
(a) Runtime on augmented BAFU dataset. **(b) Runtime on augmented Soccer dataset.**

Figure 12: Efficiency with increasing the number of series on augmented datasets.

Finally, we evaluate the impact of the anticipatory termination and the incremental initialization on ORBITS's performance. We measure the efficiency of different sign vector strategies and report the average runtime by varying the sequence length and number. We consider the following algorithms: SSV (classical sign vector), SSVi (SSV with incremental initialization), ASV (anticipatory sign vector with 1s as initialization), and ASVi (ASV with incremental initialization). Figure 13 depicts the results.

We observe that ASV has a substantial impact on ORBITS's performance and improves its runtime on average by 120x compared to using SSV (cf. Figures 13a and 13b). The incremental initialization further improves SSV and ASV by 1.7x and 1.3x, respectively.

We observe similar results when varying the missing rate. The good performance of ASV is attributed to its ability to leverage two inherent properties of time series data: the spatial continuity (similarity between time series) and the temporal continuity (similarity between the existing data and the appended one). In the unlikely case where the two conditions do not hold at the same time, ASV would not early terminate and would require the same number of iterations as SSV. The incremental initialization would, however, still improve the runtime of ORBITS by up to 2x.



(a) Varying sequence length. **(b) Varying sequence number.**

Figure 13: AVG runtime of different sign vector strategies.

7 CONCLUSIONS

In this paper, we introduced an incremental matrix decomposition technique to recover missing blocks in time series streams. Our algorithm leverages the matrix similarity before and after streams update to reduce the runtime complexity from quadratic to linear. The complexity reduction does not affect the recovery accuracy, as our technique computes an exact decomposition.

We reimplemented all the well-known online recovery algorithms using the same framework and subjected them to a broad mix of real-world time series datasets and recovery scenarios. The empirical evaluation demonstrates that our incremental algorithm significantly outperforms the state of the art both in accuracy and efficiency. Also, we provided proofs that show the scalability and the correctness of our algorithm.

In future work, we plan to investigate a holistic approach to repair time series. Another promising direction is to investigate novel parallelizable recovery techniques that use distributed frameworks.

ACKNOWLEDGMENTS

This work was supported by the Swiss National (grant agreement 169840/ProvDS). Additional funding was provided by the European Research Council (ERC) under the European Union Horizon 2020 Research and Innovation Programme (grant agreement 683253/Graphint).

We thank Jonas Traub for giving us access to the soccer dataset. We would like also to give special thanks to the anonymous reviewers for their insightful comments and suggestions.

APPENDIX

A. Matrix Similarity

Proof. Let $\mathbf{0}$ be a matrix of zeros, $\mathbf{Y}$ be an $(n+r)\times m$ matrix consisting of $\mathbf{X}$ appended with $\mathbf{0}_{r\times m}$ and let $\mathbf{M}$ be an $(n+r)\times m$ matrix consisting of $\mathbf{0}_{n\times m}$ incremented with $\Delta\mathbf{X}$. Then, we have :

$$\widetilde{\mathbf{X}} = \begin{bmatrix} \mathbf{X} \\ \Delta\mathbf{X} \end{bmatrix} = \begin{bmatrix} \mathbf{X} \\ \mathbf{0}_{r\times m} \end{bmatrix} + \begin{bmatrix} \mathbf{0}_{n\times m} \\ \Delta\mathbf{X} \end{bmatrix}$$
$$= \begin{bmatrix} \mathbf{X} \\ 0_{1*} \\ \vdots \\ 0_{r*} \end{bmatrix} + \begin{bmatrix} \mathbf{0}_{n\times m} \\ A_{1*} \\ \vdots \\ A_{r*} \end{bmatrix}$$
$$= \mathbf{Y} + \mathbf{M} \tag{12}$$

By transposing both sides of (12), multiplying each side by $\tilde{Z}$ and applying the l-2 norm, we get

$$\|\widetilde{\mathbf{X}}^T\cdot\tilde{Z}\| = \|(\mathbf{Y}+\mathbf{M})^T\cdot\tilde{Z}\|$$
$$= \|\mathbf{Y}^T\cdot\tilde{Z} + \mathbf{M}^T\cdot\tilde{Z}\| \tag{13}$$

The computation of the right hand side of (13) gives

$$\mathbf{Y}^T\cdot\tilde{Z} = \begin{bmatrix}\mathbf{X}^T 0_{1*}^T \cdots 0_{r*}^T\end{bmatrix}\cdot\begin{bmatrix} Z \\ \tilde{z}_{n+1} \\ \vdots \\ \tilde{z}_{n+r}\end{bmatrix}$$
$$= \mathbf{X}^T\cdot Z + \tilde{z}_{n+1}\times 0_{1*}^T + \cdots + \tilde{z}_{n+r}\times 0_{r*}^T$$
$$= \mathbf{X}^T\cdot Z \tag{14}$$

and

$$\mathbf{M}^T\cdot\tilde{Z} = \begin{bmatrix} \mathbf{0}_{n\times m} \\ A_{1*} \\ \vdots \\ A_{r*}\end{bmatrix}^T \cdot \begin{bmatrix} \tilde{z}_1 \\ \vdots \\ \tilde{z}_{n+r}\end{bmatrix}$$
$$= \sum_{i=1}^{n} \tilde{z}_i \times 0_{i*} + \tilde{z}_{n+1}\times A_{1*}^T + \cdots + \tilde{z}_{n+r}\times A_{r*}^T$$
$$= \tilde{z}_{n+1}\times A_{1*}^T + \cdots + \tilde{z}_{n+r}\times A_{r*}^T . \tag{15}$$

Putting (14) and (15) into (13) gives

$$\|\widetilde{\mathbf{X}}^T\cdot\tilde{Z}\| = \|\mathbf{X}^T\cdot Z + \tilde{z}_{n+1}\times A_{i*}^T + \cdots + \tilde{z}_{n+r}\times A_{r*}^T\| \tag{16}$$

Equation (16) is valid for all $\tilde{Z}$ and Z including the maximizing one, i.e., Z_{max}. It follows:

$$\|\widetilde{\mathbf{X}}^T\cdot\tilde{Z}_{max}\| = \|\mathbf{X}^T\cdot Z_{max} + \tilde{z}_{n+1}\times A_{i*}^T + \cdots + \tilde{z}_{n+r}\times A_{r*}^T\| \tag{17}$$

The application of time series temporal continuity to (17) gives

$$\|\widetilde{\mathbf{X}}^T\cdot\tilde{Z}_{max}\| = \|\mathbf{X}^T\cdot Z_{max}\| + \sum_{i=1}^{r}\|\tilde{z}_{n+i}\times A_{i*}^T\| \tag{18}$$

Since $\tilde{Z}_{max}$ is the maximizing vector of $\|\widetilde{\mathbf{X}}^T\cdot\tilde{Z}\|$, and Z_{max} is the maximizing sign vector of $\|\mathbf{X}^T\cdot Z\|$, then we get

$$\max\|\widetilde{\mathbf{X}}^T\cdot\tilde{Z}\| = \max\|\mathbf{X}^T\cdot Z\| + \sum_{i=1}^{r}\|\tilde{z}_{n+i}\times A_{i*}^T\|$$

We have $z_{n+i} = \pm 1$, hence $\forall i \in \{1,\cdots,r\}$, $\|\tilde{z}_{n+i}\times A_{i*}^T\| = \|A_{i*}^T\|$ and $\|A_{i*}^T\| = \|A_{i*}\|$. Therefore, we have

$$\max\|\widetilde{\mathbf{X}}^T\cdot\tilde{Z}\| = \max\|\mathbf{X}^T\cdot Z\| + \sum_{i=1}^{r}\|A_{i*}\|$$

which concludes the proof. □

REFERENCES

[1] [n. d.]. BundesAmt Für Umwelt – Swiss Federal Office for the Environment. https://www.hydrodaten.admin.ch/en. Last accessed: Februray 1, 2020..

[2] Orly Alter, Patrick O. Brown, and David Botstein. 2000. Singular value decomposition for genome-wide expression data processing and modeling. *Proceedings of the National Academy of Sciences* 97, 18 (2000), 10101–10106. https://doi.org/10.1073/pnas.97.18.10101 arXiv:https://www.pnas.org/content/97/18/10101.full.pdf

[3] Oren Anava, Elad Hazan, and Assaf Zeevi. 2015. Online Time Series Prediction with Missing Data. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015 (JMLR Workshop and Conference Proceedings)*, Francis R. Bach and David M. Blei (Eds.), Vol. 37. JMLR.org, 2191–2199. http://proceedings.mlr.press/v37/anava15.html

[4] Ines Arous, Mourad Khayati, Philippe Cudré-Mauroux, Ying Zhang, Martin L. Kersten, and Svetlin Stalinlov. 2019. RecovDB: Accurate and Efficient Missing Blocks Recovery for Large Time Series. In *35th IEEE International Conference on Data Engineering, ICDE 2019, Macao, China, April 8-11, 2019*. IEEE, 1976–1979. https://doi.org/10.1109/ICDE.2019.00218

[5] Laura Balzano, Yuejie Chi, and Yue M. Lu. 2018. Streaming PCA and Subspace Tracking: The Missing Data Case. *Proc. IEEE* 106, 8 (2018), 1293–1310. https://doi.org/10.1109/JPROC.2018.2847041

[6] Jason R. Belvins and Moody T. Chu. 2004. Updating the Centroid Decomposition with Applications in LSI, http://jblevins.org/research/centroid/.

[7] Laure Berti-Équille, Hazar Harmouch, Felix Naumann, Noël Novelli, and Saravanan Thirumuruganathan. 2018. Discovery of Genuine Functional Dependencies from Relational Data with Missing Values. *Proc. VLDB Endow.* 11, 8 (2018), 880–892. https://doi.org/10.14778/3204028.3204032

[8] Matthew Brand. 2002. Incremental Singular Value Decomposition of Uncertain Data with Missing Values. In *Computer Vision - ECCV 2002, 7th European Conference on Computer Vision, Copenhagen, Denmark, May 28-31, 2002, Proceedings, Part I (Lecture Notes in Computer Science)*, Anders Heyden, Gunnar Sparr, Mads Nielsen, and Peter Johansen (Eds.), Vol. 2350. Springer, 707–720. https://doi.org/10.1007/3-540-47969-4_47

[9] José Cambronero, John K. Feser, Micah J. Smith, and Samuel Madden. 2017. Query Optimization for Dynamic Imputation. *Proc. VLDB Endow.* 10, 11 (2017), 1310–1321. https://doi.org/10.14778/3137628.3137641

[10] Moody T. Chu and Robert Funderlic. 2002. The Centroid Decomposition: Relationships between Discrete Variational Decompositions and SVDs. *SIAM J. Matrix Analysis Applications* 23, 4 (2002), 1025–1044.

[11] Çagatay Demiralp, Peter J. Haas, Srinivasan Parthasarathy, and Tejaswini Pedapati. 2017. Foresight: Recommending Visual Insights. *PVLDB* 10, 12 (2017), 1937–1940. https://doi.org/10.14778/3137765.3137813

[12] Germain Forestier, François Petitjean, Hoang Anh Dau, Geoffrey I. Webb, and Eamonn J. Keogh. 2017. Generating Synthetic Time Series to Augment Sparse Datasets. In *2017 IEEE International Conference on Data Mining, ICDM 2017, New Orleans, LA, USA, November 18-21, 2017*. 865–870. https://doi.org/10.1109/ICDM.2017.106

[13] Dimitrios Giouroukis, Julius Hülsmann, Janis von Bleichert, Morgan Geldenhuys, Tim Stullich, Felipe Oliveira Gutierrez, Jonas Traub, Kaustubh Beedkar, and Volker Markl. 2019. Resense: Transparent Record and Replay of Sensor Data in the Internet of Things. In *Advances in Database Technology - 22nd International Conference on Extending Database Technology, EDBT 2019, Lisbon, Portugal, March 26-29, 2019*. OpenProceedings.org, 590–593. https://doi.org/10.5441/002/edbt.2019.63

[14] Anders Haug, Frederik Zachariassen, and Dennis van Liempd. 2011. The costs of poor data quality. *Journal of Industrial Engineering and Management* 4, 2 (2011), 168–193. https://doi.org/10.3926/jiem..v4n2.p168-193

[15] Moongu Jeon, Haesun Park, and J. Ben Rosen. 2001. Dimension reduction based on centroids and least squares for efficient processing of text data. In *Proceedings of the First SIAM International Conference on Data Mining, SDM 2001, Chicago, IL, USA, 2001*. 1–13. https://doi.org/10.1137/1.9781611972719.21

[16] Ian Jolliffe. 2002. *Principal component analysis*. Springer Verlag, New York.

[17] Kosmas Karadimitriou and John M. Tyler. 1998. The Centroid method for compressing sets of similar images. *Pattern Recognition Letters* 19, 7 (1998), 585–593.

[18] Ryan Kennedy, Laura Balzano, Stephen J. Wright, and Camillo J. Taylor. 2016. Online algorithms for factorization-based structure from motion. *Computer Vision and Image Understanding* 150 (2016), 139–152. https://doi.org/10.1016/j.cviu.2016.04.011

[19] Mourad Khayati, Michael H. Böhlen, and Philippe Cudré-Mauroux. 2015. Using Lowly Correlated Time Series to Recover Missing Values in Time Series: A Comparison Between SVD and CD. In *Advances in Spatial and Temporal Databases - 14th International Symposium, SSTD 2015, Hong Kong, China, 2015. Proceedings*. 237–254. https://doi.org/10.1007/978-3-319-22363-6_13

[20] Mourad Khayati, Michael H. Böhlen, and Johann Gamper. 2014. Memory-efficient centroid decomposition for long time series. In *IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014*. IEEE Computer Society, 100–111. https://doi.org/10.1109/ICDE.2014.6816643

[21] Mourad Khayati, Philippe Cudré-Mauroux, and Michael H. Böhlen. 2020. Scalable recovery of missing blocks in time series with high and low cross-correlations. *Knowl. Inf. Syst.* 62, 6 (2020), 2257–2280. https://doi.org/10.1007/s10115-019-01421-7

[22] Mourad Khayati, Alberto Lerner, Zakhar Tymchenko, and Philippe Cudré-Mauroux. 2020. Mind the Gap: An Experimental Evaluation of Imputation of Missing Values Techniques in Time Series. *Proc. VLDB Endow.* 13, 5 (2020), 768–782. http://www.vldb.org/pvldb/vol13/p768-khayati.pdf

[23] Marius Kloft and Pavel Laskov. 2012. Security analysis of online centroid anomaly detection. *J. Mach. Learn. Res.* 13 (2012), 3681–3724. http://dl.acm.org/citation.cfm?id=2503359

[24] Mohammad Malekzadeh, Richard G. Clegg, Andrea Cavallaro, and Hamed Haddadi. 2019. Mobile Sensor Data Anonymization. In *Proceedings of the International Conference on Internet of Things Design and Implementation (IoTDI '19)*. ACM, New York, NY, USA, 49–58. https://doi.org/10.1145/3302505.3310068

[25] Bryan David Minor, Janardhan Rao Doppa, and Diane J. Cook. 2017. Learning Activity Predictors from Sensor Data: Algorithms, Evaluation, and Applications. *IEEE TKDE.* 29, 12 (2017), 2744–2757. https://doi.org/10.1109/TKDE.2017.2750669

[26] Ioannis Mitliagkas, Constantine Caramanis, and Prateek Jain. 2013. Memory Limited, Streaming PCA. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*. 2886–2894. http://papers.nips.cc/paper/5035-memory-limited-streaming-pca

[27] Ioannis Mitliagkas, Constantine Caramanis, and Prateek Jain. 2014. Streaming PCA with Many Missing Entries. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[28] Christopher Mutschler, Holger Ziekow, and Zbigniew Jerzak. 2013. The DEBS 2013 grand challenge. In *debs, 2013*. 289–294.

[29] Milos Nikolic, Mohammed Elseidy, and Christoph Koch. 2014. LINVIEW: incremental view maintenance for complex analytical queries. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, Curtis E. Dyreson, Feifei Li, and M. Tamer Özsu (Eds.). ACM, 253–264. https://doi.org/10.1145/2588555.2610519

[30] Spiros Papadimitriou, Jimeng Sun, and Christos Faloutsos. 2005. Streaming Pattern Discovery in Multiple Time-Series. In *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*, Klemens Böhm, Christian S. Jensen, Laura M. Haas, Martin L. Kersten, Per-Åke Larson, and Beng Chin Ooi (Eds.). ACM, 697–708. http://www.vldb.org/archives/website/2005/program/paper/thu/p697-papadimitriou.pdf

[31] Spiros Papadimitriou, Jimeng Sun, Christos Faloutsos, and Philip S. Yu. 2013. Dimensionality Reduction and Filtering on Time Series Sensor Streams. In *Managing and Mining Sensor Data*. 103–141. https://doi.org/10.1007/978-1-4614-6309-2_5

[32] Irene Rodriguez-Lujan, Jordi Fonollosa, Alexander Vergara, Margie Homer, and Ramon Huerta. 2014. On the calibration of sensor arrays for pattern recognition using the minimal number of experiments. *Chemometrics and Intelligent Laboratory Systems* 130 (2014), 123 – 134.

[33] Conrad Sanderson and Ryan R. Curtin. 2018. A User-Friendly Hybrid Sparse Matrix Class in C++. In *Mathematical Software - ICMS 2018 - 6th International Conference, South Bend, IN, USA, July 24-27, 2018, Proceedings (Lecture Notes in Computer Science)*, James H. Davenport, Manuel Kauers, George Labahn, and Josef Urban (Eds.), Vol. 10931. Springer, 422–430. https://doi.org/10.1007/978-3-319-96418-8_50

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

[35] Luan Tran, Liyue Fan, and Cyrus Shahabi. 2016. Distance-based Outlier Detection in Data Streams. *PVLDB* 9, 12 (2016), 1089–1100.

[36] Kevin Wellenzohn, Michael H. Böhlen, Anton Dignös, Johann Gamper, and Hannes Mitterer. 2017. Continuous Imputation of Missing Values in Streams of Pattern-Determining Time Series. In *Proceedings of the 20th International Conference on Extending Database Technology, EDBT 2017, Venice, Italy, March 21-24, 2017*. OpenProceedings.org, 330–341. https://doi.org/10.5441/002/edbt.2017.30

[37] Xiuwen Yi, Yu Zheng, Junbo Zhang, and Tianrui Li. 2016. ST-MVL: Filling Missing Values in Geo-Sensory Time Series Data. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, Subbarao Kambhampati (Ed.). IJCAI/AAAI Press, 2704–2710. http://www.ijcai.org/Abstract/16/384

[38] Jinsung Yoon, William R. Zame, and Mihaela van der Schaar. 2019. Estimating Missing Data in Temporal Data Streams Using Multi-Directional Recurrent Neural Networks. *IEEE Trans. Biomed. Engineering* 66, 5 (2019), 1477–1490. https://doi.org/10.1109/TBME.2018.2874712

[39] Aoqian Zhang, Shaoxu Song, Jianmin Wang, and Philip S. Yu. 2017. Time Series Data Cleaning: From Anomaly Detection to Anomaly Repairing. *Proc. VLDB Endow.* 10, 10 (2017), 1046–1057. https://doi.org/10.14778/3115404.3115410

[40] Dejiao Zhang and Laura Balzano. 2016. Global Convergence of a Grassmannian Gradient Descent Algorithm for Subspace Estimation. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics AISTATS, Cadiz, Spain, 2016*. 1460–1468. http://jmlr.org/proceedings/papers/v51/zhang16b.html