Knowledge and Information Systems
https://doi.org/10.1007/s10115-019-01421-7

REGULAR PAPER

Scalable recovery of missing blocks in time series with high and low cross-correlations

Mourad Khayati[1] · Philippe Cudré-Mauroux[1] · Michael H. Böhlen[2]

Received: 26 November 2018 / Revised: 14 October 2019 / Accepted: 26 October 2019

Abstract
Missing values are very common in real-world data including time-series data. Failures in power, communication or storage can leave occasional blocks of data missing in multiple series, affecting not only real-time monitoring but also compromising the quality of data analysis. Traditional recovery (imputation) techniques often leverage the correlation across time series to recover missing blocks in multiple series. These recovery techniques, however, assume high correlation and fall short in recovering missing blocks when the series exhibit variations in correlation. In this paper, we introduce a novel approach called CDRec to recover large missing blocks in time series with high and low correlations. CDRec relies on the centroid decomposition (CD) technique to recover multiple time series at a time. We also propose and analyze a new algorithm called Incremental Scalable Sign Vector to efficiently compute CD in long time series. We empirically evaluate the accuracy and the efficiency of our recovery technique on several real-world datasets that represent a broad range of applications. The results show that our recovery is orders of magnitude faster than the most accurate algorithm while producing superior results in terms of recovery.

Keywords Recovery of missing blocks · Time series · Centroid decomposition · Correlation

Mourad Khayati received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 732328 (FashionBrain). Philippe Cudré-Mauroux received funding from the European Research Council (ERC) under the European Union Horizon 2020 Research and Innovation Programme (Grant Agreement 683253/Graphint).

✉ Mourad Khayati
mkhayati@exascale.info; mourad.khayati@unifr.ch

Philippe Cudré-Mauroux
philippe.cudre-mauroux@unifr.ch

Michael H. Böhlen
boehlen@ifi.uzh.ch

[1] eXascale Infolab, University of Fribourg, Fribourg, Switzerland

[2] Department of Informatics, University of Zurich, Zurich, Switzerland

Published online: 15 November 2019

1 Introduction

Time-series data can be found in nearly every domain, for example, climate, traffic, finance, industry and medicine. In such fields, missing values often occur (e.g., the Intel Berkeley research laboratory dataset [7] is missing about 50%, and the UCI repository of time series [11] is missing about 20%). Missing values often appear consecutively, forming a block in a time series. Some missing blocks can be rather large, because, for instance, it can take minutes, hours or even days for a broken sensor to be replaced. Data management systems assume no such gaps exist in the data. Even if a system can work with incomplete data (e.g., NULLs in relational databases), leaving missing values untreated can cause incorrect or ill-defined results [8].

The recovery of missing values has been extensively studied in the literature. Several techniques have been proposed to recover missing values, but only a few of them are able to handle large missing blocks (see Sect. 2). Existing block recovery algorithms often leverage the correlation across time series. These techniques, however, fall short when time series exhibiting variations in correlation are used. This limits the recovery accuracy since using both highly and lowly correlated time series can be beneficial to the recovery process [16].

In this paper, we study the problem of the recovery of missing blocks in multiple univariate time series[1] exhibiting variations in correlation. More specifically, we introduce a new matrix-based algorithm called CDRec that accurately recovers missing blocks in highly and lowly correlated time series. Unlike standard matrix completion techniques [23,25], our technique embeds the time-series cross-correlation into its optimization problem and thus makes it possible to take into account the variation in correlation.

At the technical level, our recovery technique relies on the centroid decomposition (CD) technique. The latter decomposes an $n \times m$ input matrix $\mathbf{X}$ into the product of two matrices, $\mathbf{X} = \mathbf{L} \cdot \mathbf{R}^{\mathrm{T}}$, where $\mathbf{L}$ and $\mathbf{R}$ are called the *loading* and *relevance* matrix, respectively, and $\mathbf{R}^{\mathrm{T}}$ denotes the transpose of $\mathbf{R}$. Each loading and relevance column (vector) is determined based on a maximal centroid value, $\max \|\mathbf{X}^{\mathrm{T}} \cdot Z\|$, which is equal to the norm of the product between the transpose of the input matrix and a sign vector Z consisting of 1s and -1s. Finding the *maximizing sign vector* that maximizes the centroid value is at the core of the CD method. The most efficient algorithm to compute the maximizing sign vector [9] requires the construction of a correlation matrix with a quadratic space complexity. This high complexity hinders the application of CD to recover long time series.

To solve the scalability problem of CD, we introduce a new algorithm called Incremental Scalable Sign Vector *(ISSV)* that efficiently computes the maximizing sign vector for an $n \times m$ input matrix, $\mathbf{X}$, that represents m time series with n observations each. Compared to the technique introduced in [9], our proposed solution does not require the construction of a correlation matrix and thus reduces space. Compared to our earlier technique introduced in [17], our proposed solution computes the weight vectors in an incremental fashion and thus speeds up the computation.

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

- We introduce a new parameter-free algorithm based on the centroid decomposition technique to recover large missing blocks in multiple time series. Our algorithm is able to handle time series with high variations in correlation.
- We propose a new sign vector computation algorithm, called Incremental Scalable Sign Vector (ISSV), that reduces the space complexity of the centroid decomposition technique from quadratic to linear and improves its runtime by an order of magnitude.

[1] We consider time series with equally spaced granularity.

– We present the results of an experimental evaluation of the recovery accuracy and efficiency of CDRec, and the efficiency and the correctness of ISSV on real-world time series. The results show that CDRec is orders of magnitude faster than the most accurate algorithm while producing superior results in terms of recovery.
– We reimplement most of the missing-block recovery techniques in a common language (C++) and make public the source code and the datasets.[2]

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 provides preliminary concepts and definitions. Section 4 describes the CDRec algorithm to recover missing blocks in multiple time series. Section 5 introduces the ISSV algorithm to compute the sign vector used by CD in linear space complexity and discusses its properties. Section 6 reports the results of our experiments. Section 7 concludes the paper and points to future work.

2 Related work

2.1 Centroid decomposition

The centroid decomposition (CD) is a matrix decomposition technique that computes the loading and the relevance vectors out of an input matrix $\mathbf{X}$. The most challenging part of CD is the computation of the sign vector Z, consisting of 1s and -1s, that maximizes $\|\mathbf{X}^{\mathrm{T}} \cdot Z\|$, where $\mathbf{X}^{\mathrm{T}}$ is the transpose of $\mathbf{X}$ and $\|\cdot\|$ denotes the norm of a vector. The classical approach is based on the centroid method [10] with a brute-force search through an exponential number of sign vectors (see Sect. 3.3). This yields an exponential time and linear space complexity. Chu et al. [9] introduce a more efficient algorithm to find the maximizing sign vector, which we refer to as *Quadratic Sign Vector* (QSV). The authors consider the set of all possible sign vectors 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 the search through a traversal along the nodes of the hypercube. QSV achieves a quadratic runtime complexity, and its space complexity is quadratic too because of the construction of the correlation matrix. In [17], we introduce the SSV algorithm that computes the sign vectors without the construction of the covariance/correlation matrix and achieves a linear space complexity. 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 needs to be flipped. The weight vector is obtained through mapping the original optimization problem that CD solves onto a different and equivalent optimization problem.

This work extends our earlier results [17] as follows. First, we introduce the CDRec algorithm that uses CD to recover missing blocks in time series and describe its properties (cf. Sect. 4). Second, we introduce the ISSV technique to incrementally compute the sign vectors, yielding a faster CD computation compared to the SSV technique (cf. Sect. 5). Next, we prove the correctness of our flipping strategy which guarantees an optimal recovery (cf. Sect. 5.3). Finally, we empirically compare the recovery accuracy of CDRec against the state-of-the-art recovery techniques we describe below (cf. Sect. 6).

[2] Source code and datasets are available online: https://github.com/eXascaleInfolab/2019_kais-bench.git.

2.2 Missing-blocks recovery techniques

Several statistical techniques have been proposed in the literature to recover missing values such as Regression [13], MeanImpute [14] and kNNImpute [33]. Data analysis tools, such as R package, implement a wide range of these statistical techniques.[3] These techniques are, however, effective only in the case of single or a handful of missing values and are not suitable in the case of large missing blocks [22,34], which is the main focus of this paper. In what follows, we describe three different categories of techniques designed to recover missing blocks in time-series data.

Matrix-based techniques They recover missing blocks by looking at an entire set of series as a matrix and by applying techniques based on matrix completion principles. These algorithms rely on different matrix decomposition/factorization techniques such as principal components analysis, matrix factorization, nonnegative matrix factorization and singular spectrum analysis.

In [32], the authors propose the TRMF technique that uses matrix factorization (MF) to recover missing blocks in multidimensional time series. MF takes an $n \times m$ input matrix $\mathbf{X}$ and approximates it using two factor matrices, $\mathbf{W}$ and $\mathbf{H}$, respectively, of size $n \times r$ and $r \times m$ such that $\mathbf{X} \approx \mathbf{W} \cdot \mathbf{H}$. The resulting factorization is embedded with a new temporal autoregressive regularizer to learn the dependencies across the input time series. The temporal dependencies are then used to infer replacement values for the missing ones. TRMF resorts to an autoregressive model, which makes it not scalable for large time series.

Mei et al. [19] propose a temporal nonnegative matrix factorization (NMF)-based technique called TeNMF to recover missing blocks. NMF is similar to the above MF technique, but it constrains $\mathbf{W}$ and $\mathbf{H}$ to contain nonnegative elements only. The authors formalize the matrix recovery problem as a minimization of a quadratic nonnegative loss function of the difference between $\mathbf{V}$ and the product $\mathbf{W} \cdot \mathbf{H}$. Then, a penalization term is introduced into the loss function to take into account for the cross-correlation between time series. TeNMF's recovery is sensitive to negative correlations as it resorts to NMF principles.

Balzano et al. [5,6] propose a principal component analysis (PCA)-based recovery technique called GROUSE. PCA takes an $n \times m$ input matrix $\mathbf{X}$ and finds n components each of size m that approximate the dimensions of the initial data. GROUSE applies an incremental gradient descent procedure on a defined cost function to track the co-evolving dimensions and subsequently derive the missing values. The proposed technique does not initialize the missing values rendering GROUSE's recovery unstable.

In [2,3], the authors introduce a recovery algorithm that relies on the singular spectrum analysis (SSA) technique. The proposed technique takes the input time series and constructs a so-called page matrix (a non-overlapping Hankel matrix[4]). Then, the singular value decomposition (SVD) [21] is applied to the page matrix in order to extract the singular values. The latter are grouped depending on the model that generates the time series and used to approximate the original matrix. The approximated matrix is used as a source of replacement for the missing values. This technique does not support multiple incomplete time series.

Pattern-based techniques These techniques consider that sensors which are at close proximity can present trend similarity. They apply pattern matching techniques and use the observed values as a source of replacement.

[3] https://cran.r-project.org/web/views/MissingData.html.

[4] https://en.wikipedia.org/wiki/Hankel_matrix.

TKCM [29] is a continuous technique to recover missing blocks in correlated time-series streams. It first defines a pattern as interval of points, across all time series, that contains the missing value. Then, it searches for the k most similar non-overlapping patterns to the defined pattern and returns the average of points over the k patterns as an estimation of the missing value. TKCM is able to handle linear and nonlinear correlated time series and performs a scalable recovery that is linear with the length of time series. TKCM is designed for time series with repeating trends and is not capable of recovering multiple time series at a time.

Yi et al. [30] introduce the STMVL technique to recover missing blocks in geosensory time series. The proposed technique leverages the temporal (closeness of the values in time) and spatial (distance between time series) correlation between time series. STMVL combines a user-based and item-based collaborative filtering with statistical smoothing models to derive models out of the historical data. The missing values are then estimated using the generated models and the spatio-temporal coordinates of values. STMVL assumes the input time series to be highly correlated.

Network-based techniques These algorithms build a (parametric) model which reconstructs the linear dependence between time series. The recovery is based on information obtained through those dependencies.

Yoon et al. [31] introduce a neural network (NN)-based solution called MRNN to recover missing blocks. The proposed solution first initializes the missing values using linear interpolation and then applies a multi-directional recurrent network to recover the missing data. MRNN learns the data dependencies by leveraging both the correlation within time series and the correlation across time series. MRNN was designed for medical data where time series are dependent on one another.

In [18], the authors introduce a deep network-based recovery technique called DeepIN. The proposed technique is designed for a smart city environment where the missing blocks follow a repeating pattern. DeepIN uses multiple long short-term memory (LSTM) models to learn the temporal-spatial dependencies across time series and assumes the time series to be highly correlated.

In Sect. 6, we compare the efficiency and accuracy of CDRec against all the aforementioned recovery, except DeepIN, for which no source code is publicly available. Our results show that, in addition to be parameter-free, our recovery outperforms the state of the art.

3 Background

3.1 Notations and definitions

Bold uppercase letters refer to matrices, regular font uppercase letters refer to vectors (rows and columns of matrices), and lowercase letters refer to elements of vectors/matrices. For example, $\mathbf{X}$ is matrix, X_{i*} is the ith row of $\mathbf{X}$, X_{*i} is the ith column of $\mathbf{X}$, $(X_{i*})^{\mathrm{T}}$ is the transpose of X_{i*}, and x_{ij} is the jth element of X_{i*}. The isolated column 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 an ordered set of n temporal values v_i that are ordered according to their time stamps t_i. In the rest of the paper, we omit the time stamps, since they are ordered, and write the time series $X_1 = \{(0, 2), (1, 0), (2, -4)\}$ as the ordered set $X_1 = \{2, 0, -4\}$. 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 the symbol $\times$ for scalar multiplications and the symbol $\cdot$ for matrix multiplications. The symbol $\|\|$ refers to the l-2 norm of a vector. Assume $X = [x_1, \ldots, x_n]$, then $\|X\| = \sqrt{\sum_{i=1}^{n}(x_i)^2}$.

3.2 Centroid Decomposition

The *Centroid Decomposition (CD)* 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.,

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

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

The function CD describes the centroid decomposition procedure of an $n \times m$ input matrix $\mathbf{X}$. At each iteration i, the *maximizing sign vector* Z (described in Sect. 3.3) is computed and used to subsequently compute the ith relevance and loading vectors. Next, a matrix reduction step is applied in order to obtain the next relevance and loading vectors. The algorithm terminates when m loading and relevance vectors, of size n and m, respectively, are computed. Note that the m maximizing sign vectors are different and independent from each other.

function CD($\mathbf{X}$, n, m)
  $i := 1$
  repeat
    $Z_i := FindMaxSV(\mathbf{X}, n, m)$
    $C_i := \|\mathbf{X}^{\mathrm{T}} \cdot Z_i\|$
    $R_i := \frac{C_i}{\|C_i\|}$
    $L_i := \mathbf{X} \cdot R_i$
    $\mathbf{X} := \mathbf{X} - L_i \cdot R_i^{\mathrm{T}}$
    $i := i + 1$
  until $i = m$;
  return $\mathbf{L}$, $\mathbf{R}$
end function

Example 1 (Centroid Decomposition) To illustrate the computation of CD, consider the input matrix, $\mathbf{X}$, that contains three time series of five elements each as a running example, i.e.,

$$\mathbf{X} = \begin{bmatrix} 6 & 3 & 3 \\ -2 & 2 & 2 \\ -7 & 1 & -5 \\ -3 & 4 & -1 \\ 2 & -4 & 2 \end{bmatrix}$$

Among all sign vectors, the sign vector that maximizes $\|\mathbf{X}^{\mathrm{T}}\cdot Z\|$ is $Z_1 = \{-1, 1, 1, 1, -1\}^{\mathrm{T}}$. Z_1 is used to compute the first column of $\mathbf{R}$ (and $\mathbf{L}$) during iteration 1 as follows:

$$R_{*1}=\frac{\mathbf{X}^{\mathrm{T}}\cdot Z_1}{\|\mathbf{X}^{\mathrm{T}}\cdot Z_1\|}=\begin{bmatrix}0.86\\0.19\\0.48\end{bmatrix};\quad L_{*1}=\mathbf{X}\cdot R_{*1}=\begin{bmatrix}5.27\\-1.63\\-8.27\\-4.33\\3.86\end{bmatrix}$$

Similarly, the second and third columns of $\mathbf{R}$ (and $\mathbf{L}$) are computed using the second and third maximizing sign vectors (derived from $\mathbf{X}-L_{*1}\cdot R_{*1}^{\mathrm{T}}$), respectively. The resulting decomposition produced by CD is (only two decimals are shown):

$$\mathbf{X}=\begin{bmatrix}6&3&3\\-2&2&2\\-7&1&-5\\-3&4&-1\\2&-4&2\end{bmatrix}=\underbrace{\begin{bmatrix}5.27&5&1.1\\-1.63&2.19&-2.14\\-8.27&-2.33&1.1\\-4.33&2.67&0.41\\3.86&-2.48&-1.73\end{bmatrix}}_{\mathbf{L}}\cdot\underbrace{\begin{bmatrix}0.86&-0.34&0.39\\0.19&0.91&0.38\\0.48&0.25&-0.84\end{bmatrix}}_{\mathbf{R}^{\mathrm{T}}}$$

3.3 Maximizing sign vector

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

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

To illustrate the computation of the maximizing sign vector, consider the same input matrix from our running example. We proceed by enumerating all possible sign vectors, and we compute $\|\mathbf{X}^{\mathrm{T}}\cdot Z\|$ for each of them. (The computed values are displayed below the sign vectors.)

Z_1	Z_2	Z_3	Z_4	Z_5	Z_6	Z_7	$\cdots$	Z_{10}	$\cdots$	Z_{32}
$\begin{bmatrix}1\\1\\1\\1\\1\end{bmatrix}$	$\begin{bmatrix}-1\\1\\1\\1\\1\end{bmatrix}$	$\begin{bmatrix}1\\-1\\1\\1\\1\end{bmatrix}$	$\begin{bmatrix}1\\1\\-1\\1\\1\end{bmatrix}$	$\begin{bmatrix}1\\1\\1\\-1\\1\end{bmatrix}$	$\begin{bmatrix}1\\1\\1\\1\\-1\end{bmatrix}$	$\begin{bmatrix}-1\\-1\\1\\1\\1\end{bmatrix}$	$\cdots$	$\begin{bmatrix}-1\\1\\1\\1\\-1\end{bmatrix}$	$\cdots$	$\begin{bmatrix}-1\\-1\\-1\\-1\\-1\end{bmatrix}$
7.2	16.7	3.6	15.3	4.1	16.4	15.5		23.3		7.2

Among all sign vectors, Z_{10} and its sign opposite give the same and maximum centroid value $\|\mathbf{X}^{\mathrm{T}}\cdot Z\| = 23.3$, and it is thus the maximizing sign vector. Assume that $diag^{=0}(\mathbf{X})$ is an auxiliary function that sets the diagonal values of an $n\times n$ matrix to 0, then according to Lemma 1 in [17], the following equivalence holds:

$$\underset{Z\in\{1,-1\}^n}{\arg\max}\ \|\mathbf{X}^{\mathrm{T}}\cdot Z\| \equiv \underset{Z\in\{1,-1\}^n}{\arg\max}\ Z^{\mathrm{T}}\cdot V \tag{2}$$

where $V = diag^{=0}\left(\mathbf{X}\cdot\mathbf{X}^{\mathrm{T}}\right)\cdot Z$. The elements of V are defined as follows:

$$v_i = z_i(z_i\times X_{i*}\cdot S-(X_{i*}\cdot(X_{i*})^{\mathrm{T}})) \tag{3}$$

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

In the next section, we describe the CDRec algorithm that uses CD to recover missing blocks in time series.

4 Recovery of missing blocks

4.1 Recovery process

Algorithm 1 uses CD to recover missing blocks in multiple time series at a time. It takes as input a matrix $\mathbf{X}$ that contains a set of missing blocks $\mathbf{X}_B$ and a list T^- of pairs indicating the rows and columns of the missing values in $\mathbf{X}$. We normalize the data using the z-score normalization technique [15]. The recovery starts by initializing $\mathbf{X}_B$ using either linear interpolation or extrapolation, depending on the position of the missing block(s) in $\mathbf{X}$ (line 2). Then, we apply a truncation to the decomposition of $\mathbf{X}$ to return $\mathbf{L}_k$ and $\mathbf{R}_k$ which contain the first k columns of $\mathbf{L}$ and $\mathbf{R}$, respectively (line 6). To dynamically set, at each iteration, the truncation factor k, we utilize the commonly used entropy method (described later). Next, the values in $\mathbf{X}$ with positions in T^- are updated with their corresponding ones in $\mathbf{X}_k = \mathbf{L}_k \cdot \mathbf{R}_k^{\mathrm{T}}$ (lines 8–9). On line 10, we cache the computed sign vector and we use it as an initial sign vector in the CD computation of the next iteration of the recovery. The recovery process terminates if the relative difference in Frobenius norm $\|\mathbf{X}_B - \widetilde{\mathbf{X}}_B\|_F/|B| = \sqrt{\sum_{i=1}^{|B|}(\tilde{x}_i - \tilde{x}_i')^2}/|B|$ between $\widetilde{\mathbf{X}}_B$ and $\mathbf{X}_B$ (where $\tilde{x}_i \in \mathbf{X}$, $\tilde{x}_i \in \widetilde{\mathbf{X}}$ and $|B|$ is the length of the missing block) falls below a small threshold value ϵ (by default 10^{-5}).

Algorithm 1: CDRec($\mathbf{X}$, T^-)

Input : $n \times m$ matrix $\mathbf{X}$; List of missing time points T^-
Output: Matrix with recovered values $\widetilde{\mathbf{X}}$

1 Linearly interpolate/extrapolate all missing values in $\mathbf{X}$;
2 $Z_{init} := [1, \ldots, 1]$;
3 **repeat**
4 $\quad \widetilde{\mathbf{X}} := \mathbf{X}$;
5 $\quad$ compute truncation factor k of $\mathbf{X}$ ▷ using Entropy-based technique
6 $\quad [\mathbf{L}_k, \mathbf{R}_k, Z] := CD(\mathbf{X}, n, k, Z_{init})$;
7 $\quad \mathbf{X}_k := \mathbf{L}_k \cdot (\mathbf{R}_k)^T$;
$\quad$ `// Update missing values`
8 $\quad$ **foreach** $(i, j) \in T^-$ **do**
9 $\quad\quad \tilde{x}_{ij} := y_{ij}$;
$\quad\quad$ `//` y_{ij} `element of` $\mathbf{X}_k$ `at timestamp` i
10 $\quad Z_{init} := Z$ ▷ cache the sign vector
11 **until** $\frac{\|\mathbf{X}_B - \widetilde{\mathbf{X}}_B\|_F}{|B|} < \epsilon$;
12 **return** $\widetilde{\mathbf{X}}$;

Example 2 Let's take the example of the input matrix $\mathbf{X}$ from our running example (cf. Example 1). We illustrate the application of CDRec to recover a missing block (in gray) in the first time series.

$$\underbrace{\begin{bmatrix} 6 & 3 & 3 \\ -2 & 2 & 2 \\ -7 & 1 & -5 \\ -3 & 4 & -1 \\ 2 & -4 & 2 \end{bmatrix}}_{\mathbf{X}} \rightarrow \underbrace{\begin{bmatrix} 6 & 3 & 3 \\ -2 & 2 & 2 \\ - & 1 & -5 \\ - & 4 & -1 \\ 2 & -4 & 2 \end{bmatrix}}_{\mathbf{X}_{miss}} \rightarrow \underbrace{\begin{bmatrix} 6 & 3 & 3 \\ -2 & 2 & 2 \\ -0.66 & 1 & -5 \\ 0.66 & 4 & -1 \\ 2 & -4 & 2 \end{bmatrix}}_{\mathbf{X}_{init}} \rightarrow \underbrace{\begin{bmatrix} 6 & 3 & 3 \\ -2 & 2 & 2 \\ -5.1 & 1 & -5 \\ -1.06 & 4 & -1 \\ 2 & -4 & 2 \end{bmatrix}}_{\widetilde{\mathbf{X}}_1} \rightarrow \cdots \rightarrow \underbrace{\begin{bmatrix} 6 & 3 & 3 \\ -2 & 2 & 2 \\ -6.98 & 1 & -5 \\ -4 & 4 & -1 \\ 2 & -4 & 2 \end{bmatrix}}_{\widetilde{\mathbf{X}}_p}$$

First, the missing block in $\mathbf{X}_{miss}$ is initialized and the resulting matrix, $\mathbf{X}_{init}$, is decomposed using a truncation factor $k = 1$ to produce $\mathbf{L}_1$ and $\mathbf{R}_1^{\mathrm{T}}$. By multiplying $\mathbf{L}_1$ and $\mathbf{R}_1^{\mathrm{T}}$, we obtain $\widetilde{\mathbf{X}}_1$. After applying the same process p times, the final $\widetilde{\mathbf{X}}_p$ containing the recovered values is obtained. Using longer time series helps to improve the accuracy of the recovery as described in Sect. 6.

To dynamically set the truncation factor k, we use an entropy-based method. Let $\mathbf{X}$ be an input matrix of n rows and m columns with $n \gg m$, and let $f_k = \frac{\|\mathbf{X}^{\mathrm{T}} \cdot Z_k\|}{\sum_{i=1}^{m} (\|\mathbf{X}^{\mathrm{T}} \cdot Z_k\|)^2}$ be the relative contribution of the kth centroid value to the Frobenius norm. Then, the entropy of the centroid values is defined as follows:

$$E = -\frac{1}{\log m} \sum_{i=1}^{m} f_i \log f_i \tag{4}$$

The entropy E is used to find the appropriate truncation factor for the recovery process. More specifically, at each iteration of Algorithm 1, we select the smallest k such that $\sum_{i=1}^{k} f_i \geq E$. Since the entropy (used to find the truncation value k) minimizes the Frobenius norm [4] as does the iterative process, our recovery quickly converges (cf. Sect. 6.2.3).

4.2 Recovery properties

The following lemma shows that the recovery of CDRec embeds the correlation across the input time series.

Lemma 1 *Let* $\mathbf{X}$ *be an* $n \times m$ *input matrix containing* m *time series where some of them have missing blocks. Then, the recovery of CDRec is based on the summation of the time-series cross-correlation.*

Proof From (2), we have

$$V = diag^{=0}\left(\mathbf{X} \cdot \mathbf{X}^{\mathrm{T}}\right) \cdot Z = diag^{=0}\left(\begin{bmatrix} x_{11} \ldots x_{1n} \\ x_{21} \ldots x_{2n} \\ \vdots \\ x_{m1} \ldots x_{mn} \end{bmatrix} \cdot \begin{bmatrix} x_{11} & x_{21} \ldots x_{m1} \\ & \vdots \\ x_{1n} & x_{2n} \ldots x_{mn} \end{bmatrix}\right) \cdot Z$$

Let $\bar{x}_{*i}$ and σ_{*i} be, respectively, the mean and the standard deviation of X_{*i}. Since each column of $\mathbf{X}$ is z-score normalized, it follows

$$V = diag^{=0}\left(\begin{bmatrix} \frac{x_{11}-\bar{x}_{*1}}{\sigma_{*1}} & \cdots & \frac{x_{1n}-\bar{x}_{*n}}{\sigma_{*n}} \\ \frac{x_{21}-\bar{x}_{*1}}{\sigma_{*1}} & \cdots & \frac{x_{2n}-\bar{x}_{*n}}{\sigma_{*n}} \\ & \vdots & \\ \frac{x_{21}-\bar{x}_{*1}}{\sigma_{*1}} & \cdots & \frac{x_{2n}-\bar{x}_{*n}}{\sigma_{*n}} \end{bmatrix} \cdot \begin{bmatrix} \frac{x_{11}-x_{*1}}{\sigma_{*1}} & \frac{x_{21}-x_{*1}}{\sigma_{*1}} & \cdots & \frac{x_{21}-x_{*1}}{\sigma_{*1}} \\ & & \vdots & \\ \frac{x_{1n}-x_{*n}}{\sigma_{*n}} & \frac{x_{2n}-x_{*n}}{\sigma_{*n}} & \cdots & \frac{x_{2n}-x_{*n}}{\sigma_{*n}} \end{bmatrix}\right) \cdot Z \tag{5}$$

Consider the product of the first row of $\mathbf{X}$ and the first column of $\mathbf{X}^{\mathrm{T}}$. we have

$$\begin{bmatrix} \frac{x_{11}-\bar{x}_{*1}}{\sigma_{*1}} & \cdots & \frac{x_{1n}-\bar{x}_{*n}}{\sigma_{*n}} \end{bmatrix} \cdot \begin{bmatrix} \frac{x_{11}-\bar{x}_{*1}}{\sigma_{*1}} \\ \vdots \\ \frac{x_{1n}-\bar{x}_{*n}}{\sigma_{*n}} \end{bmatrix} = \frac{\sum_{i=1}^{n}(x_{1i}-\bar{x}_{*i})(x_{1i}-\bar{x}_{*i})}{\sigma_{*1}\sigma_{*1}}$$
$$= r_{11} \quad (r \text{ is the Pearson correlation}) \tag{6}$$

Putting (6) into (5), we get

$$V = diag^{=0}\left(\begin{bmatrix} r_{11} & r_{12}\ldots r_{1n} \\ r_{21} & r_{22}\ldots r_{2n} \\ & \vdots & \\ r_{m1} & r_{m2}\ldots r_{mn} \end{bmatrix}\right) \cdot Z = \begin{bmatrix} 0 & r_{12}\ldots r_{1n} \\ r_{21} & 0\ldots r_{2n} \\ & \vdots & \\ r_{m1} & r_{m2}\ldots 0 \end{bmatrix} \cdot Z = \begin{bmatrix} \sum_{i=1}^{n}(z_i \times r_{1i}) - z_1 \times r_{11} \\ \sum_{i=1}^{n}(z_i \times r_{1i}) - z_2 \times r_{22} \\ \vdots \\ \sum_{i=1}^{n}(z_i \times r_{1i}) - z_n \times r_{nn} \end{bmatrix}$$

The elements of V used for the decomposition (and subsequently the recovery) are computed by summing up the cross-correlation between the columns of $\mathbf{X}$ which concludes the proof.

5 Incremental Scalable Sign Vector (ISSV)

In this section, we first introduce a new incremental algorithm called ISSV to incrementally compute the maximizing sign vector in linear space. Then, we discuss the properties of the ISSV algorithm.

5.1 Incremental weight vector

Lemma 2 (Weight vectors are incremental) *Let $Z^{(k)}$ be Z at iteration k, p the position of the last element flipped in $Z^{(k)}$ and v_i the ith value in V. For any two consecutive iterations of sign vectors, the weight vectors are linearly dependent, i.e.,*

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

Proof By definition of the weight vector (cf. (2)), we have

$$V^{(k)} = diag^{=0}\left(\mathbf{X}\cdot\mathbf{X}^{\mathrm{T}}\right) \cdot Z^{(k)}$$
$$V^{(k+1)} = diag^{=0}\left(\mathbf{X}\cdot\mathbf{X}^{\mathrm{T}}\right) \cdot Z^{(k+1)} \tag{7}$$

Let p be the index of the element in $Z^{(k)}$ that has been flipped, and let U_p be the unit vector with the same length as $Z^{(k)}$ where the pth element is 1 and all other elements are 0. Using U_p, we compute $Z^{(k+1)}$ as follows

$$Z^{(k+1)} = Z^{(k)} - 2 \times U_p \tag{8}$$

Putting (8) into (7), we get

$$\begin{aligned} V^{(k+1)} &= diag^{=0}\left(\mathbf{X}{\cdot}\mathbf{X}^{\mathrm{T}}\right) \cdot (Z^{(k)} - 2 \times U_p) \\ &= diag^{=0}\left(\mathbf{X}{\cdot}\mathbf{X}^{\mathrm{T}}\right) \cdot Z^{(k)} - 2 \times diag^{=0}\left(\mathbf{X}{\cdot}\mathbf{X}^{\mathrm{T}}\right) \cdot U \\ &= V^{(k)} - 2 \times diag^{=0}\left(\mathbf{X}{\cdot}\mathbf{X}^{\mathrm{T}}\right) \cdot U_p \end{aligned} \tag{9}$$

Let $col(\mathbf{X}, p)$ return the pth column of $\mathbf{X}$. Then, from (9), we get

$$\begin{aligned} V^{(k+1)} &= V^{(k)} - 2 \times col(diag^{=0}\left(\mathbf{X}{\cdot}\mathbf{X}^{\mathrm{T}}\right), p) \\ &= V^{(k)} - 2 \times \begin{bmatrix} X_{1*} \cdot X_{p*}^T \\ X_{2*} \cdot X_{p*}^T \\ \vdots \\ X_{n*} \cdot X_{p*}^T \end{bmatrix} \end{aligned}$$

Thus, $\forall i \in [1, n] \setminus \{p\}$; we have

$$v_i^{(k+1)} = v_i^{(k)} - 2 \times (X_{i*} \cdot X_{p*}^T) \tag{10}$$

Lemma 2 allows us to incrementally compute $V^{(k+1)}$ out of $V^{(k)}$ and subsequently to incrementally compute the maximizing sign vector. Unlike the iterative approach [17], the incremental approach does not require the construction of the intermediate vectors S and Z each of length n yielding faster computation. We compare the efficiency of both approaches in Sect. 6.

We propose an algorithm with linear space complexity to incrementally compute the maximizing sign vector Z according to the maximization problem introduced in (2). The basic idea is as follows. We begin with the sign vector $Z = [1, \ldots, 1]^{\mathrm{T}}$, change the sign of the element in Z that increases $Z^{\mathrm{T}} \cdot V$ most and incrementally compute the weight vector. The algorithm terminates when Z and V have the same pairwise sign.

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

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

```
1 Z^T := [1, ..., 1];
2 V := Compute initial weight vector                    ▷ using (3)
3 repeat
4     p := {i | v_i = min(v_j ∈ V) & z_j × v_j < 0};
5     z_p := (−1) × z_p;
6     foreach v_i ∈ V \ v_p do
7         v_i := v_i − 2 × X_i* · (X_p*)^T
8 until p = 0;
9 return Z ∈ [1, −1]^n;
```

Algorithm 2 implements the incremental strategy to compute the maximizing sign vector. We note that V is computed directly from $\mathbf{X}$ by reading the matrix row by row, one row at a time to compute the individual elements of V. We first compute the initial weight vector according to (3). Then, we search for the index (p) 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 flipped. A new vector V is incrementally computed out of the previous one, 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 and thus, i.e., p is equal to 0. The vector Z in the final iteration is the maximizing sign vector that maximizes $Z^{\mathrm{T}} \cdot V$.

Example 3 To illustrate the computation of the sign vector using Algorithm 2, consider the same input matrix as the one introduced in our running example. We denote $Z^{(k)}$ as Z in the kth iteration. First, Z is initialized with 1s, i.e., $Z^{(1)} = \{1, 1, 1, 1, 1\}^{\mathrm{T}}$, and the initial weight vector is computed using (3) to get $V^{(1)} = \{-57, 10, -46, 9, -54\}^{\mathrm{T}}$. All elements of $Z^{(1)}$ have a different sign from the corresponding elements in $V^{(1)}$, and among them the element in the first position has the highest absolute value. Using $p = 1$, the next weight vector is incrementally computed (using (10)) as follows

$$
\begin{aligned}
v_1 &= -57 \\
v_2 &= 10 - 2 \times \left(\begin{bmatrix} 6 & 3 & 3 \end{bmatrix} \times \begin{bmatrix} -2 \\ 2 \\ 2 \end{bmatrix} \right) = 10 \\
v_3 &= -46 - 2 \times \left(\begin{bmatrix} 6 & 3 & 3 \end{bmatrix} \times \begin{bmatrix} -7 \\ 1 \\ -5 \end{bmatrix} \right) = 62 \\
v_4 &= 9 - 2 \times \left(\begin{bmatrix} 6 & 3 & 3 \end{bmatrix} \times \begin{bmatrix} -3 \\ 4 \\ -1 \end{bmatrix} \right) = 27 \\
v_5 &= -54 - 2 \times \left(\begin{bmatrix} 6 & 3 & 3 \end{bmatrix} \times \begin{bmatrix} 2 \\ -4 \\ 2 \end{bmatrix} \right) = -66
\end{aligned}
$$

i.e.,

$$
Z^{(2)} = \begin{bmatrix} -1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad \text{and} \quad V^{(2)} = \begin{bmatrix} -57 \\ 10 \\ 62 \\ 27 \\ -66 \end{bmatrix}.
$$

Only the fifth element in $Z^{(2)}$ has a different sign from the corresponding element in $V^{(2)}$, In the third iteration of the algorithm, we flip the sign of the element at position 5 in $Z^{(2)}$ and we use the new sign vector $Z^{(3)}$ to compute $V^{(3)}$, similar to the previous iteration, and get

$$
Z^{(3)} = \begin{bmatrix} -1 \\ 1 \\ 1 \\ 1 \\ -1 \end{bmatrix} \text{ and } V^{(3)} = \begin{bmatrix} -69 \\ 26 \\ 118 \\ 75 \\ -66 \end{bmatrix}
$$

Since all corresponding elements in $Z^{(3)}$ and $V^{(3)}$ have the same sign, the algorithm terminates and $Z^{(3)}$ is returned as the maximizing sign vector that maximizes $Z^{\mathrm{T}} \cdot V$. Note that our technique computes the same sign vector as the one computed by the exhaustive search illustrated in Sect. 3.3.

Example 3 illustrates the main properties of the ISSV algorithm. First, the product $Z^{\mathrm{T}} \cdot V$ is monotonically increasing at each iteration, i.e., $(Z^{(1)})^{\mathrm{T}} \cdot V^{(1)} = -138$, $(Z^{(2)})^{\mathrm{T}} \cdot V^{(2)} = 90$ and $(Z^{(3)})^{\mathrm{T}} \cdot V^{(3)} = 354$. Second, the algorithm terminates and computes the global optimum, i.e., 23.3.

The ISSV algorithm keeps in memory the sign vector V and one row of $\mathbf{X}^{\mathrm{T}}$, each with $O(n)$ space complexity, where n is the number of rows in $\mathbf{X}$. To compute the individual elements $v_i = v_i - 2 \times X_{i*} \cdot (X_{p*})^{\mathrm{T}}$ of the weight vector, $\mathbf{X}$ is read one row at a time. The result vector has length n; thus, the total space complexity is $O(n)$. The total runtime complexity is $O(xn)$, where x is the number of flipped elements in the returned sign vector Z. The number of flipped elements depends on the number of negative rows in the input matrix and is on average less than third of the number of rows (n) (cf. the experiment of Fig. 10 in [17]).

5.2 ISSV properties

In this section, we describe the main properties of the ISSV algorithm. Since ISSV uses the same definition of the weight vector as the SSV algorithm, the two algorithms share the same properties. More specifically, the computation of ISSV is monotonic, terminates and returns the correct result.

Let $Z^{(k)}$ and $V^{(k)}$ refer to, respectively, vectors V and Z in the kth iteration of the ISSV algorithm. $v_i^{(k)}$ and $z_i^{(k)}$ denote, respectively, the ith element of $V^{(k)}$ and $Z^{(k)}$. Lemma 3 shows that the computation of the maximizing sign vector in the ISSV algorithm is *strictly monotonic*, i.e., each iteration increases the value of $Z^{\mathrm{T}} \cdot V$.

Lemma 3 (Monotonicity) *For any two consecutive iterations k and $k+1$ in the ISSV algorithm, the following holds:*

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

Lemma 4 shows that ISSV does not flip a sign value more than once and thus terminates.

Lemma 4 (Termination) *Let $\mathbf{X}$ be an $n \times m$ matrix. ISSV($\mathbf{X}$, n, m) terminates and performs at most n iterations.*

Lemma 5 shows that our greedy approach computes the optimal solution.

Lemma 5 (Correctness) *The ISSV algorithm computes the maximizing sign vector for which the final product $Z^{\mathrm{T}} \cdot V$ is the global maximum.*

The proofs of Lemma 3, 4 and 5 are described in detail in [17].

5.3 Flipping strategy

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

Proof We do a case analysis to prove that ISSV makes the local optimal choice.

Case $z_i^{(k)} \times v_i^{(k)} < 0$: The ISSV algorithm changes in each iteration the sign of an element $z_i^{(k)}$ that maximizes $|z_i^{(k)} \times v_i^{(k)}|$. Since $diag^{=0}\left(\mathbf{X}\cdot\mathbf{X}^{\mathrm{T}}\right)$ is a symmetric matrix, we have

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

Without loss of generality, we assume $z_1^{(k)} \times v_1^{(k)} < 0$, $z_n^{(k)} \times v_n^{(k)} < 0$ such that $|v_1^{(k)}| > |v_n^{(k)}|$. Let Z_1 and Z_n be the sign vectors resulting from changing the sign of z_1 and z_n, respectively. Then, we have

$$\begin{aligned} (Z_1)^{\mathrm{T}} \cdot V^{(k)} &> (Z_n)^{\mathrm{T}} \cdot V^{(k)} \\ (Z_1)^{\mathrm{T}} \cdot (diag^{=0}\left(\mathbf{X}\cdot\mathbf{X}^{\mathrm{T}}\right) \cdot Z^{(k)}) &> (Z_n)^{\mathrm{T}} \cdot (diag^{=0}\left(\mathbf{X}\cdot\mathbf{X}^{\mathrm{T}}\right) \cdot Z^{(k)}) \\ \sum_{i=2}^{n-1} x_{in} &> \sum_{i=2}^{n-1} x_{1i} \end{aligned} \tag{11}$$

Let's assume that we get a bigger benefit by changing the sign of z_n instead of z_1. Then, we get

$$\begin{aligned} (Z_1)^{\mathrm{T}} \cdot V_1^{(k+1)} &< (Z_n)^{\mathrm{T}} \cdot V_n^{(k+1)} \\ (Z_1)^{\mathrm{T}} \cdot (diag^{=0}\left(\mathbf{X}\cdot\mathbf{X}^{\mathrm{T}}\right) \cdot Z_1) &< (Z_n)^{\mathrm{T}} \cdot (diag^{=0}\left(\mathbf{X}\cdot\mathbf{X}^{\mathrm{T}}\right) \cdot Z_n) \\ \sum_{i=2}^{n-1} x_{in} &< \sum_{i=2}^{n-1} x_{1i} \end{aligned}$$

This contradicts Equation (11). Therefore, we get a bigger benefit by choosing the element that maximizes $|z_i^{(k)} \times v_i^{(k)}|$, i.e., z_1.

Case $z_i^{(k)} \times v_i^{(k)} \geq 0$: If instead we changed 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 3), which implies $(Z^{(k+1)})^{\mathrm{T}} \cdot V^{(k+1)} \leq (Z^{(k)})^{\mathrm{T}} \cdot V^{(k)}$. Therefore, $Z^{\mathrm{T}} \cdot V$ will not be increased.

We now show that two alternative sign flipping strategies to compute the sign vector do not produce the global maximum.

Strategy 1: change at each iteration the sign of more than one element in Z. The maximization problem looks as follows:

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

The result of the maximization is obtained first by multiplying the ith element of $(Z^{(k)})^{\mathrm{T}}$ with the ith column of $diag^{=0}\left(\mathbf{X}\cdot\mathbf{X}^{\mathrm{T}}\right)$, yielding a vector. Then, the ith element of the resulting vector is multiplied again with the ith element of $Z^{(k)}$. Thus, changing the sign of $z_i^{(k)}$ affects only the ith column of $diag^{=0}\left(\mathbf{X}\cdot\mathbf{X}^{\mathrm{T}}\right)$; all other columns are unaffected. Thus, the result of changing $z_i^{(k)}$ is independent from changing $z_j^{(k)}$ for $j \neq i$. Since we are maximizing over independent variables, the maximization of $(Z^{(k)})^{\mathrm{T}} \cdot V^{(k)}$ is obtained by checking the result of changing one element in $Z^{(k)}$ at a time and not if more than one element is changed.

Strategy 2: change at each iteration the sign of an element $z_i^{(k)}$ for which $z_i^{(k)} \times v_i^{(k)} > 0$. Following the same reasoning as in the proof of Lemma 3, we have $Z^{(k+1)} = Z^{(k)} + 2 \times U$, which implies that $(Z^{(k+1)})^{\mathrm{T}} \cdot V^{(k+1)} < (Z^{(k)})^{\mathrm{T}} \cdot V^{(k)}$. Therefore, this strategy is strictly monotonically decreasing, and thus, the global maximum will not be obtained.

6 Experimental evaluation

This section evaluates in turn the performance of i) our CDRec (recovery using CD) algorithm and ii) the ISSV algorithm that efficiently computes CD. For each experiment, we report the average result over five consecutive runs of the various algorithms.

6.1 Setup

6.1.1 Implementation

We compare CDRec against the state-of-the-art missing-blocks recovery techniques: SSA [2], GROUSE [5], TeNMF [19], TKCM [29], STMVL [30], MRNN [31] and TRMF [32]. We rewrote all these algorithms in C++, except SSA and MRNN (for which we use the efficient original implementations). We use the same advanced linear algebra operations across all techniques, thanks to a modern library called Armadillo [27]. The source code and the datasets are available online.[5]

All the following experiments were performed on a Linux machine with a 3.4 GHz Intel Core i7 and 16GB of RAM. The code was compiled with g++ 7.3.0 at the maximum optimization level.

6.1.2 Datasets

The following empirical evaluation was performed on various real-world datasets containing time series of different (Pearson) correlation values. Each tuple in the time series records a time stamp and the value of an observation. The values of the observations are stored as four-byte floating numbers, while the sign vectors are binary arrays. The *BAFU* dataset, kindly provided by the BundesAmt Für Umwelt (the Swiss Federal Office for the Environment) [12], contains water discharge time series collected from different Swiss rivers containing between 200k and 1.3 million values each and covers the time period from 1974 to 2015. The *MeteoSwiss* dataset, kindly provided by the Swiss Federal Office of Meteorology and Climatology [20], contains weather time series recorded in different cities in Switzerland. Also, we use publicly available real-world time series: statistical quality control on *Baseball*

[5] https://github.com/eXascaleInfolab/2019_kais-bench.git.

Table 1 Description of time series

Name	TS max_length	nb of TS	Granularity	Correlation (r)
BAFU TS	1.3M	12	30 min	[−0.03, 0.89]
MeteoSwiss TS	200k	7	10 min	[−0.12,0.9]
Baseball TS	2k	4	1 year	[−0.53, 0.49]
Temperature TS	19k	12	1 day	[0.78, 0.89]
Gas	3600	24	6 h	[−0.75,0.78]

time series [24], *Temperature* values of 703 climate stations in China collected from 1960 to 2012 [1] and *Gas* concentration batches that were gathered between 2007 and 2011 in a gas delivery platform facility situated at the ChemoSignals Laboratory at UCSD [26,28]. For the gas dataset, we choose the longest batches to which we refer to as Gas6 and Gas10.

The datasets we use in our experiments represent a broad range of applications. They contain time series which exhibit different levels of correlations that are representative of many aspects naturally present in real-world data. Table 1 describes the main properties of our time series.

6.2 Recovery evaluation

In this section, we compare the accuracy and the efficiency of CDRec against the aforementioned techniques under different recovery scenarios.

6.2.1 Accuracy

In Table 2, we evaluate the accuracy of all the techniques using all datasets from Table 1. We set the length and the number of the time series, respectively, to their maximum per dataset. We first evaluate the case where the missing block appears in one time series. Then, we evaluate the case where the missing block appears in multiple time series. As accuracy measure, we use the root-mean-square error (RMSE) between the original blocks and the recovered ones.

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

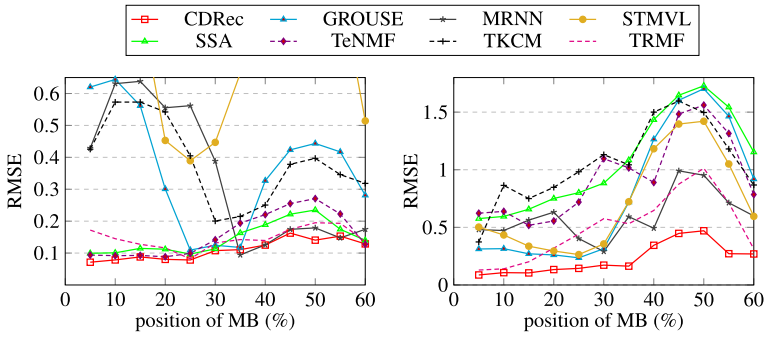
where T is the set of missing values, and x_t and $\tilde{x}_t$ are the original and the recovered values, respectively.

On the left-hand side of Table 2, we set a missing block to appear arbitrarily in the middle of one of the series in the dataset. We then vary the size of the missing block from 20 to 80% of the chosen series and measure the recovery accuracy using RMSE. The results show that CDRec is on average 2.3× more accurate than the most accurate state-of-the-art algorithm, TRMF. Thanks to the weight vectors that embed the correlation across the input time series, CDRec is able to better leverage the similarity between time series and to accurately recover the missing blocks. The superiority of CDRec is more visible on datasets with high variations of correlation, e.g., Baseball and Gas datasets. We also observe that the error of CDRec does not always increase along with the size of the missing block in this experiment. In three

Table 2 Recovery accuracy on real-word datasets

	Methods	Varying % of miss. val.				Varying # of incomplete TS		
		20%	40%	60%	80%	2	3	4
Baseball.	**CDRec**	**0.47**	**0.48**	**0.46**	**0.43**	**0.50**	**0.50**	0.92
	GROUSE	2.30	2.72	3.49	4.40	5.7	5.7	4.96
	MRNN	0.54	0.70	1.36	0.46	0.96	1.13	0.95
	SSA	0.47	0.54	0.50	0.51	–	–	–
	STMVL	1.05	1.08	1.06	1.05	1.10	0.97	1.01
	TeNMF	0.68	0.80	3.39	0.68	0.95	0.98	0.89
	TKCM	1.20	1.20	1.18	1.13	–	–	–
	TRMF	0.58	0.66	0.76	0.84	0.60	0.58	**0.77**
MeteoSwiss	**CDRec**	**0.05**	**0.04**	**0.03**	**0.05**	**0.04**	**0.09**	**0.15**
	GROUSE	5.35	4.99	4.25	4.32	5.90	4.86	4.80
	MRNN	0.31	0.42	0.62	0.51	0.35	0.42	0.48
	SSA	0.10	0.09	0.10	0.13	–	–	–
	STMVL	0.32	0.28	0.25	0.26	0.25	0.26	0.63
	TeNMF	0.07	0.08	0.08	0.31	0.09	0.13	0.83
	TKCM	0.13	0.14	0.27	0.40	–	–	–
	TRMF	0.05	0.05	0.04	0.15	0.04	0.23	0.28
Gas	**CDRec**	**0.08**	**0.1**	**0.23**	0.61	**0.56**	**0.58**	**0.55**
	GROUSE	0.31	0.27	1.01	0.98	0.52	2.87	2.70
	MRNN	0.37	0.77	0.68	**0.55**	0.79	0.74	0.95
	SSA	0.50	0.41	0.74	0.70	–	–	–
	STMVL	0.57	0.69	1.10	1.04	1.19	1.21	1.21
	TeNMF	0.62	0.61	0.81	0.91			
	TKCM	0.37	0.68	1.10	1.03	–	–	–
	TRMF	0.12	0.32	0.62	0.68	0.65	0.60	0.63
BAFU	**CDRec**	**0.07**	**0.08**	**0.15**	0.22	**0.20**	0.17	**0.16**
	GROUSE	0.62	0.48	0.60	0.54	0.47	0.42	0.38
	MRNN	0.42	0.46	0.48	0.49	0.43	0.43	0.39
	SSA	0.25	0.18	0.22	**0.20**	–	–	–
	STMVL	1.04	0.79	0.92	0.81	0.75	0.67	0.62
	TeNMF	0.09	0.10	0.29	0.39	0.18	**0.15**	0.17
	TKCM	0.42	0.48	0.47	0.52	–	–	–
	TRMF	0.17	0.14	0.42	0.47	0.24	0.20	0.18
Temp.	**CDRec**	**0.29**	**0.30**	**0.31**	**0.25**	0.28	**0.27**	**0.3**
	GROUSE	0.54	0.56	0.51	0.53	0.39	0.35	0.44
	MRNN	1.73	1.33	1.21	1.33	1.27	1.40	0.96
	SSA	0.31	0.34	0.32	0.37	–	–	–
	STMVL	0.45	0.45	0.4	0.42	0.32	0.30	0.37
	TeNMF	0.29	0.32	0.33	0.91	0.28	0.28	0.37
	TKCM	2.20	1.74	1.96	1.77	–	–	–
	TRMF	0.33	0.37	0.34	**0.33**	**0.27**	0.27	0.36

The lowest RMSE is highlighted in bold



(a) BAFU dataset **(b)** Gas dataset.

Fig. 1 Accuracy comparison with moving the missing block (MB) position

datasets, Baseball Meteo and Temperature, the trend is the opposite. In these three datasets, larger missing blocks require a larger number of iterations which, in turn, yield better recovery.

On the right-hand side of Table 2, we vary the number of incomplete time series and we measure the recovery accuracy of all techniques, except TKCM and SSA which do not support this feature. We create missing blocks with the size of 10% of the longest time series per dataset to appear completely at random in each time series. The results show that, similar to the single incomplete case, CDRec outperforms its competitors in most datasets when multiple time series are incomplete. We observe also that in some datasets, the error does not always increase along with the number of incomplete time series. As we stated earlier, the bigger the number of missing blocks, the more iterations the algorithms can perform to compute the recovery. More iterations mean further opportunities to refine the values with which the missing block was initialized.

We also evaluate the recovery accuracy when increasing the sequence length (n) and the sequence number (m). The results show similar trends as the ones depicted in Table 2. In general, all techniques take advantage of longer and/or larger number of time series to improve the recovery.

In the experiment in Fig. 1, we evaluate the impact of the position of the missing block on the recovery accuracy. We keep the length and number of time series to their maximum per dataset, and we vary the position of the missing block starting after 5% of the data. We set the size of the missing block to 10% of one time series, and we choose the BAFU and Gas datasets which contain the longest and largest number of time series, respectively.

The results show two different trends. In the BAFU dataset (cf. Fig. 1a), the accuracy of CDRec and TRMF is barely affected by the position of the missing block. For the remaining algorithms, however, the recovery drastically varies depending on the position. Most algorithms achieve their best recovery when the missing block occurs between 30 and 40% from the beginning of the data. This part of the data contains a flat trend which poses no significant recovery challenge to most of the algorithms. In the Gas dataset (cf. Fig. 1b), the accuracy of all algorithms deteriorates when the missing bock occurs between 40 and 50% from the beginning of the time series. The reason is that this part of the data contains lots of fluctuations which makes it harder for the algorithms to estimate the missing block. These results confirm that CDRec is still the best contender.

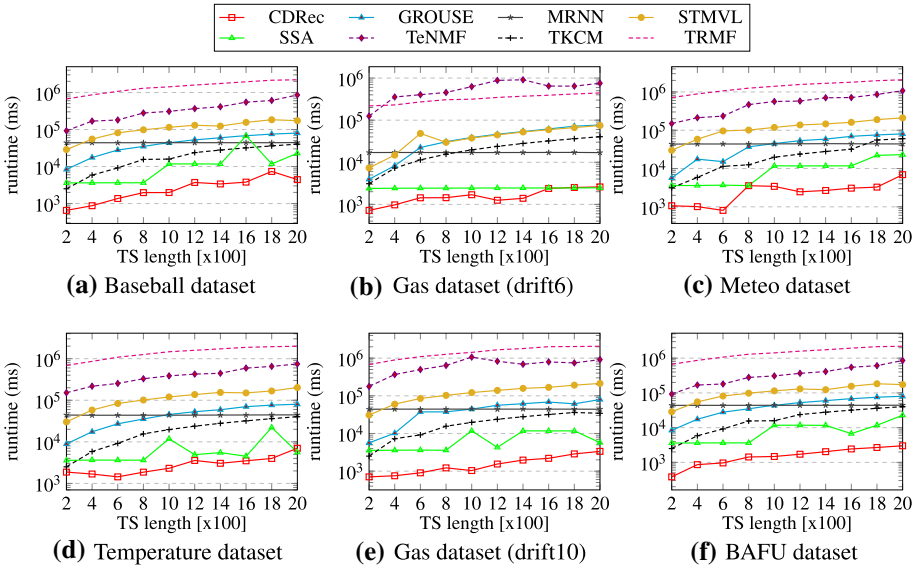


Fig. 2 Runtime varying TS length

6.2.2 Efficiency

In this section, we compare the efficiency of CDRec by varying in turn the length of time series and their number. We measure the runtime and present it on a log scale since the results vary widely across algorithms.

In the experiment in Fig. 2, we set the number of time series to the maximum number per dataset, and we vary their length between 200 and 2000 and measure the runtime (in ms) to recover a missing block of 100 observations (notice the log scale on the *y*-axis). The results of this experiment show CDRec takes on average 5 ms to recover time series containing 2000 observations each. CDRec is on average 3× faster (3ms vs 10ms) than the fastest state-of-the-art technique, SSA and two orders of magnitude faster than the most accurate state-of-the-art technique, TRMF. The efficiency of CDRec is explained by its fast incremental computation of the centroid values.

In the experiment in Fig. 3, we set the length of time series to $1k$, and we vary their number and measure the runtime (in ms) to recover a missing block of 100 observations (notice the log scale on the *y*-axis). We use the three datasets that contain the largest number of time series. The results of this experiment show that CDRec and SSA are the fastest solutions, while TRMF and TeNMF are the slowest. Thanks to its effective entropy-based truncation procedure, the runtime of CDRec barely increases while increasing the number of time series used in the recovery.

6.2.3 Convergence

In the experiments in Fig. 4, we evaluate the convergence of the CDRec's recovery by measuring the number of iterations. In Fig. 4a, we set the number of time series to the maximum number per dataset and we vary the length of time series, while in Fig. 4b, we set the length of time series to $1k$ and we vary the number of time series. The results of

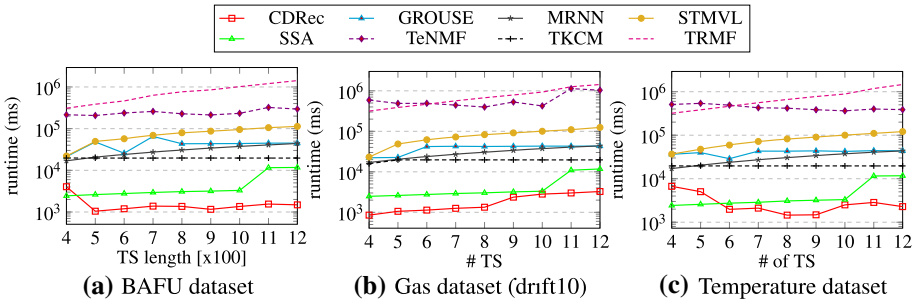


Fig. 3 Runti me varying TS number

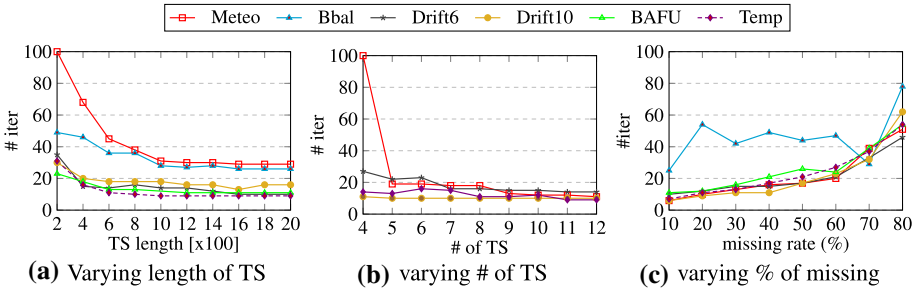


Fig. 4 Recovery Convergence

both experiments confirm that adding more observations and more time series helps CDRec reducing the number of iterations to perform the recovery until becoming constant at a specific number of observations ($1k$ values in the case of the BAFU dataset) and at a specific number of time series also (eight time series in the same dataset). In Fig. 4c, we vary the missing percentage rate in time series and we measure the number of iterations. The result of this experiment confirms that CDRec converges on all datasets, i.e., the number of iterations does not exceed 100 for different missing rates in all datasets. It is worth noticing that the relative difference of Frobenius norm computed at each iteration of Algorithm 1 decreases monotonically until reaching the specified threshold.

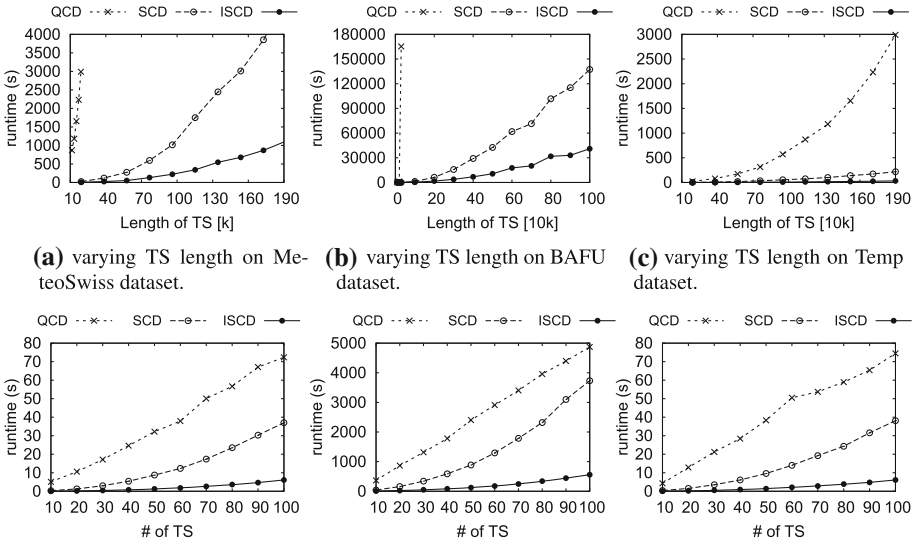
6.3 Matrix decomposition

In this section, we evaluate different techniques to compare the centroid decomposition. We compare our technique ISCD against SCD and QSV that use SSV and QSV techniques, respectively (cf. Sect. 2.1).

6.3.1 Efficiency

In order to evaluate the efficiency of our technique, we choose the three datasets with the longest time series from Table 1. Figure 5 evaluates the efficiency of our ISCD technique to decompose matrices of time series and compares it against SCD and QCD.

In the experiments shown in Fig. 5a–c, we set the number of time series to the maximum per dataset, we vary their length and we report the runtime of the three algorithms. The results of this experiment show that for the largest BAFU dataset, ISCD is more than 3× faster than



(a) varying TS length on MeteoSwiss dataset.

(b) varying TS length on BAFU dataset.

(c) varying TS length on Temp dataset.

(d) varying # of TS on MeteoSwiss dataset.

(e) varying # of TS on BAFU datset.

(f) varying # of TS on Temp dataset.

Fig. 5 Efficiency of different CD implementations

SCD. The QCD algorithm runs out of memory for $n > 10000$, as it constructs an $n \times n$ correlation matrix.

In the experiments shown in Fig. 5d–f, we set the series length to 1k, we vary their number between 10 and 100 (by splitting the series into smaller segments) and we report the runtime of the three algorithms. The results of this experiment show that all the algorithms are linear with the number of time series. The results show also that in the BAFU dataset, ISCD is up to 9× faster than QCD (9 min vs. 82 min) and 7× faster than SCD (9 min vs. 62 min), while in the two other datasets, our ISCD is up to 12× faster than QCD (6 s vs. 75 s) and 6× faster than SCD (6 s vs. 38 s).

6.3.2 Correctness

In this section, we evaluate the correctness of our technique by computing the Frobenius norm (cf. Sect. 4) between the input matrix $\mathbf{X}$ and the matrix product $\mathbf{L} \cdot \mathbf{R}^{\mathrm{T}}$ computed as a result of the decomposition. A correct CD decomposition returns $\mathbf{L}$ and $\mathbf{R}$ s.t. $\mathbf{X} = \mathbf{L} \cdot \mathbf{R}^{\mathrm{T}}$ and subsequently $FrobN = 0$. In Table 3, we vary the length of the time series while keeping their number to the maximum per dataset and compute FrobN for different CD techniques. The results show that, unlike QCD, ISCD and SCD return similar FrobN equal to zero for all datasets. Thus, both SCD and ISCD compute the correct CD decomposition, while QCD yields only an approximation of the decomposition.

7 Conclusion

In this paper, we introduced a new recovery algorithm for time series exhibiting variations in correlation. We presented the CDRec algorithm that uses the centroid decomposition

Table 3 Frobenius norm of different techniques

dataset	$FrobN$ $n = 500$			$n = 1k$			$n = 2k$		
	SCD	ISCD	QCD	SCD	ISCD	QCD	SCD	ISCD	QSV
BAFU	6.7E−15	6.5E−15	33	2.3E−14	2.4E−14	52	4.1E−14	3.9E−14	58
MeteoS	5.6E−15	5.6E−15	18	7.9E−15	7.9E−15	27	1.9E−14	1.9E−14	44
Gas6	2.4E−14	2.4E−14	56	3.3E−14	3.3E−14	73	5.7E−14	5.7E−14	92
Baseball	6.4E−15	**6.5E−15**	14	1.8E−14	1.8E−14	23	–	–	–
Temp.	1.5E−14	1.3E−14	47	2.5E−14	2.4E−14	70	3.9E−14	3.8E−14	105
Gas10	2.1E−14	2.1E−14	66	3.8E−14	4E−14	78	4.6E−14	4.6E−14	90

technique to accurately recover large missing blocks in multiple time series. We also presented an incremental approach to efficiently compute CD for long time series. We conducted experiments on several real-world time series demonstrating that our recovery technique outperforms the state of the art. The results show that CDRec is orders of magnitude faster than the most accurate algorithm while producing superior results in terms of recovery.

As future work, we plan to investigate the application of CDRec to perform continuous recovery of missing values in time-series streams.

References

1. Administration CM (2018) Temperature TS. https://www.hydrodaten.admin.ch/en. Accessed 01 July 2018
2. Agarwal A, Amjad MJ, Shah D, Shen D (2018) Model agnostic time series analysis via matrix estimation. POMACS 2(3):40:1–40:39. https://doi.org/10.1145/3287319
3. Agarwal A, Amjad MJ, Shah D, Shen D (2018) Time series analysis via matrix estimation. CoRR arXiv:1802.09064
4. Alter O, Brown PO, Botstein D (2000) Singular value decomposition for genome-wide expression data processing and modeling. Proc Natl Acad Sci USA 97(18):10101–6
5. Balzano L, Chi Y, Lu YM (2018) Streaming PCA and subspace tracking: the missing data case. In: Proceedings of the IEEE 106(7)
6. Balzano L, Nowak R, Recht B (2010) Online identification and tracking of subspaces from highly incomplete information. In: 2010 48th Annual allerton conference on communication, control, and computing (Allerton). https://doi.org/10.1109/allerton.2010.5706976
7. Bodik P, Hong W, Guestrin C, Madden S, Paskin M, Thibaux R (2004) Intel Berkeley research lab dataset. http://db.csail.mit.edu/labdata/labdata.html. Accessed 8 Nov 2018
8. Cambronero J, Feser JK, Smith MJ, Madden S (2017) Query optimization for dynamic imputation. PVLDB 10(11):1310–1321. https://doi.org/10.14778/3137628.3137641. http://www.vldb.org/pvldb/vol10/p1310-feser.pdf
9. Chu MT, Funderlic R (2002) The centroid decomposition: relationships between discrete variational decompositions and svds. SIAM J Matrix Anal Appl 23(4):1025–1044. https://doi.org/10.1137/S0895479800382555
10. D'agostino Sr, RB, Russell HK (2005) Centroid method. Wiley. https://doi.org/10.1002/0470011815.b2a13006
11. Dheeru D, Karra TE (2017) UCI machine learning repository. http://archive.ics.uci.edu/ml. Accessed 1 Jan 2019
12. for the Environment FOEN, F.O.: BAFU TS. https://www.meteoswiss.admin.ch/home.html?tab=alarm. Accessed 01 Mar 2018
13. Gold MS, Bentler PM (2000) Treatments of missing data: a monte carlo comparison of rbhdi, iterative stochastic regression imputation, and expectation-maximization. Struct Equ Model Multidiscip J 7(3):319–355. https://doi.org/10.1207/S15328007SEM0703_1

14. Hening D, Koonce DA (2014) Missing data imputation method comparison in Ohio university student retention database. In: Proceedings of the 2014 international conference on industrial engineering and operations management, Bali, Indonesia, January 7–9, 2014
15. Jain AK, Nandakumar K, Ross A (2005) Score normalization in multimodal biometric systems. Pattern Recognit 38(12):2270–2285. https://doi.org/10.1016/j.patcog.2005.01.012
16. Khayati M, Böhlen MH, Cudré-Mauroux P (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, August 26–28, 2015. Proceedings, pp. 237–254. https://doi.org/10.1007/978-3-319-22363-6_13
17. Khayati M, Böhlen MH, Gamper J (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, pp 100–111
18. Lee M, An J, Lee Y (2019) Missing-value imputation of continuous missing based on deep imputation network using correlations among multiple IOT data streams in a smart space. IEICE Trans 102-D(2):289–298. http://search.ieice.org/bin/summary.php?id=e102-d_2_289
19. Mei J, de Castro Y, Goude Y, Hébrail G (2017) Nonnegative matrix factorization for time series recovery from a few temporal aggregates. In: Proceedings of the 34th international conference on machine learning, ICML 2017, Sydney, NSW, Australia, 6–11 August 2017, pp 2382–2390
20. Meteorology FO (2018) Climatology: MeteoSwiss TS. https://www.hydrodaten.admin.ch/en. Accessed 01 Mar 2018
21. Meyer CD (2000) Matrix analysis and applied linear algebra. SIAM. https://doi.org/10.1137/1.9780898719512. https://my.siam.org/Store/Product/viewproduct/?ProductId=971
22. Moritz S, Sardá A, Bartz-Beielstein T, Zaefferer M, Stork J (2015) Comparison of different methods for univariate time series imputation in R. CoRR arXiv:1510.03924
23. Ongie G, Willett R, Nowak RD, Balzano L (2017) Algebraic variety models for high-rank matrix completion. In: Proceedings of the 34th international conference on machine learning, ICML 2017, Sydney, NSW, Australia, 6–11 August 2017, pp 2691–2700
24. Rasp J (2018) Statistical quality control. http://www.cma.gov.cn. Accessed 01 July 2018
25. Recht B (2011) A simpler approach to matrix completion. J Mach Learn Res 12:3413–3430
26. Rodriguez-Lujan I, Fonollosa J, Vergara A, Homer M, Huerta R (2014) On the calibration of sensor arrays for pattern recognition using the minimal number of experiments. Chemom Intell Lab Syst 130:123–134. https://doi.org/10.1016/j.chemolab.2013.10.012
27. Sanderson C, Curtin RR (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, pp 422–430. https://doi.org/10.1007/978-3-319-96418-8_50
28. Vergara A, Vembu S, Ayhan T, Ryan MA, Homer ML, Huerta R (2012) Chemical gas sensor drift compensation using classifier ensembles. Sens Actuat B Chem 166–167:320–329. https://doi.org/10.1016/j.snb.2012.01.074
29. Wellenzohn K, Böhlen MH, Dignös A, Gamper J, Mitterer H (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, pp 330–341
30. Yi X, Zheng Y, Zhang J, Li T (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, pp 2704–2710
31. Yoon J, Zame WR, van der Schaar M (2019) Estimating missing data in temporal data streams using multi-directional recurrent neural networks. IEEE Trans Biomed Eng 66(5):1477–1490. https://doi.org/10.1109/TBME.2018.2874712
32. Yu H, Rao N, Dhillon IS (2016) Temporal regularized matrix factorization for high-dimensional time series prediction. In: Advances in neural information processing systems 29: annual conference on neural information processing systems 2016, December 5–10, 2016, Barcelona, Spain, pp 847–855
33. Zhang S (2012) Nearest neighbor selection for iteratively KNN imputation. J Syst Softw 85(11):2541–2552. https://doi.org/10.1016/j.jss.2012.05.073
34. Zhu X (2014) Comparison of four methods for handing missing data in longitudinal data analysis through a simulation study. Open J Stat 4:933–944. https://doi.org/10.4236/ojs.2014.411088

Mourad Khayati is a Senior Researcher and Lecturer at the eXascale Infolab of University of Fribourg in Switzerland. He did his PhD at the Database Technology Group of the University of Zürich in Switzerland. He holds a BSc and MSc in Computer Science from University of Jendouba in Tunisia and INSA de Lyon in France, respectively. His main research interest is matrix decomposition techniques with a special focus on their application for the recovery of missing values in time series data.

Philippe Cudré-Mauroux is a Full Professor at the University of Fribourg, Switzerland, where he leads the eXascale Infolab. Before coming back to Switzerland, he spent a few years working with the Database Systems Lab (Sam Madden & Mike Stonbraker) at MIT. He got his BSc from EPFL and two MSc, one from Eurecom and one from INRIA SOP/U. of Sophia Antipolis. At the end of his studies, he spent six months at IBM T.J. Watson Research, working on media delivery architectures. He got his PhD from EPFL, working in Karl Aberer's lab. His dissertation on Emergent Semantics won the EPFL Doctorate Award and the EPFL Press Mention in 2007. During his PhD, he also spent some time at U.C. Berkeley working on sensor data and at Microsoft Research Asia working on novel information-sharing systems. He recently won a 2M€ ERC Grant, a Google Faculty Research Award, as well as the VeriSign Internet Infrastructures Award.

Michael H. Böhlen is the Head of the Database Technology Group at the Department of Informatics (ifi) at the University of Zürich. His research interests include various aspects of data management, and he focuses mainly on time-varying information, data warehousing, similarity search in databases, and data analysis. His work has a technological focus and includes construction of data-centric systems, query processing, data modeling and query languages, and systems architectures. Before joining the University of Zürich, he was employed at the Computer Science Departments of the Free University of Bozen-Bolzano (2003–2009), Aalborg University (1995–2003), University of Arizona (1994–1995), and ETH Zürich (1990–1994). At the Free University of Bolzano, he was the Head of the Faculty of Computer Science from August 2003 until September 2007. He was a Lecturer at the Engineering School in Grenchen, Switzerland (1990–1994).