CORAD: Correlation-Aware Compression of Massive Time Series using Sparse Dictionary Coding

Abdelouahab Khelifati
eXascale Infolab
University of Fribourg
Fribourg—Switzerland
{firstname.lastname}@unifr.ch

Mourad Khayati
eXascale Infolab
University of Fribourg
Fribourg—Switzerland
{firstname.lastname}@unifr.ch

Philippe Cudré-Mauroux
eXascale Infolab
University of Fribourg
Fribourg—Switzerland
{firstname.lastname}@unifr.ch

***Abstract*—Time series streams are ubiquitous in many application domains, e.g., transportation, network monitoring, autonomous vehicles, or the Internet of Things (IoT). Transmitting and storing large amounts of such fine-grained data is however expensive, which makes compression schemes necessary in practice. Time series streams that are transmitted together often share properties or evolve together, making them significantly correlated. Despite the rich literature on compression methods, the state-of-the-art approaches do not typically avail correlation information when compressing times series. In this work, we demonstrate how one can leverage the correlation across several related time series streams to both drastically improve the compression efficiency and reduce the accuracy loss.**

We present a novel compression algorithm for time series streams called CORAD (CORelation-Aware compression of time series streams based on sparse Dictionary coding). Based on sparse dictionary learning, CORAD has the unique ability to exploit the correlation across multiple related time series to eliminate redundancy and perform a more efficient compression. To ensure the accuracy of the compressed time series, we further introduce a method to threshold the information loss of the compression. Extensive validation on real-world datasets shows that CORAD drastically outperforms state-of-the-art approaches achieving up to 40:1 compression ratios while minimizing the information loss.

***Index Terms*—Data Compression, Time Series Streams, Correlation, IoT.**

I. Introduction

Time series streams are widely used nowadays due to the proliferation of sensor-generated data e.g., for transportation, network monitoring, autonomous vehicles, or the Internet of Things (IoT). In such applications, the data generated by sensors may get very large by involving up to millions of sensors [1]. Therefore, efficient means to store and analyze the sheer amount of resulting time series data plays an important role in practice.

Transmitting and storing large amounts of sensor data is often prohibitively expensive due to the fine-grained granularity of the transmitted data. For time series practitioners, it is hence important to consider compression schemes in order to reduce the sheer size of the data for transmission, storage but also subsequent computations on the data. This can be achieved by leveraging some of the salient features of the time series. Compression methods are traditionally organized in two broad classes: (1) lossless techniques, which allow the exact reconstruction of the original time series without any information loss, and (2) lossy techniques, which involve some information loss but typically yield significantly higher compression ratios. Sensor-generated time series are often noisy, non-uniformly sampled and misaligned in practice making an exact reconstruction of the data unnecessary. As a result, we consider lossy schemes in the following.

We focus on sets of time series emerging as multiple sequences emanating from the same source. These time series tend to be correlated as they often capture multiple facets of the same phenomenon. Take the example of the Inertial Measurement Unit (IMU) in modern smartphones, which collects multi-dimensional gyroscope, acceleration, and magnetometer data, for a total of nine variables sampled at each time step. These time series are likely to be recorded together, as each one of them taken separately is insufficient to characterize the phone's motion. At the same time, the resulting time series will be highly correlated, as the all describe the same underlying phenomenon (i.e., a smartphone moving) from different perspectives.

The characteristics of time series may change over time. In fact, this is a key property of semi-infinite streams. The correlation state between two time series may also change, resulting in local correlations which are not persistent for the entire time series (see Figure 1). Another key property of the time series we consider is that they often exhibit repeating and/or evolving *patterns*, making them a prime candidate for *sparse* dictionary learning. The idea is to segment the data and to express each segment as a weighted linear combination of only a few basic elements of the dictionary called *atoms* [2]. The atoms are learned from the data itself. Sparse coding techniques have been widely used in a number of applications ranging from video processing to texture synthesis and machine learning.

Taking advantage of both the high degree of correlation of the time series we consider as well as of their recurring nature, we introduce a new compression algorithm called CORAD (CORelation-Aware compression of time series streams based on sparse Dictionary coding). Our technique leverages both the global and local correlation across time series to eliminate redundancy for a more efficient compression. CORAD is also able to bound the information loss of the compression in an online manner and allows to execute analytical queries on the compressed data directly.

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

- We introduce CORAD to effectively compress sets of time series streams. It performs a compression that is aware of the correlation across time series. We propose a representation that is more meaningful, malleable, and robust to noise than the ones obtained by traditional methods. CORAD is, to the best of our knowledge, the first sparse compression scheme to use correlation information across time series to improve its compression efficiency;
- We introduce a method for error-bounding the resulting representation. Thanks to this method, one can bound the maximum error yielded by our scheme on time series, hence introducing a controllable trade-off between accuracy and compression ratio;
- We demonstrate the performance of our technique on several real-world time series data. We empirically show that CORAD can achieve up to 40:1 compression ratios with only minimal loss in terms of accuracy, outperforming the state-of-the-art methods.

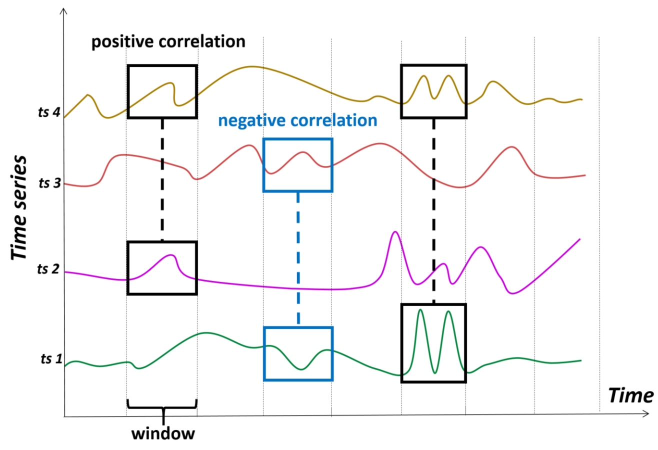


Fig. 1. Example of correlated segments in multiple time series

The rest of the paper is structured as follows. We start by surveying related work in Section II. We introduce the key concepts behind our method in Section III. We describe our method, CORAD, along with each of its sub-components in detail in Section IV. We present an extensive empirical evaluation of CORAD and related methods on real-world datasets in Section V before concluding.

II. Related Work

In a previous paper, we introduced a dictionary-based compression method called TRISTAN [2]. TRISTAN divides the time series into segments of equal sizes before coding them using a combination of atoms. It exploits sparse dictionary representations for effective compression, compact storage, and efficient query execution over the compressed data. TRISTAN can execute queries on the compressed data directly. In terms of compression efficiency, TRISTAN achieves compression ratios between 2:1 and 20:1. TRISTAN is however agnostic to the underlying correlation between sets of time series and does not leverage this property in its compression.

Other dictionary-based techniques were proposed to perform time series operations on compressed representations. For instance, the authors in [3] suggest a method that focuses on executing similarity queries on compressed time series by coding segments of time series into a dictionary. The resulting representation is a sequence of frequencies corresponding to the occurrences of the segments. Similarly, Willis et al. [4] introduce a similarity measure that uses the Limpel-Ziv dictionary-based compression scheme for time series. Along similar lines, Hu et al. [5] introduce time series classification based on dictionary-compressed time series. These techniques however do not support the reconstruction of the original data, which is a key feature in our context.

The Discrete Wavelet Transform (DWT) has been extensively used to compress time series data [6]–[9]. It uses a set of basis functions called wavelets to decompose time series into components. Time series data components are separated into different frequencies at different scales by DWT. The main purpose of DWT is to reduce the size of the data and/or to decrease its noise by providing time and frequency information. The DWT method is particularly efficient at representing time series exhibiting discontinuities or sharp peaks. It is able to achieve high compression ratios above 15:1. The DWT technique however lacks phase information, which makes it sensitive to time shifts [10]. We experimentally compare CORAD to a DWT-based approach in Section V.

Quantization algorithms [11]–[15] were also used to compress time series. They reduce multidimensional data into a smaller number of dimensions or a lower resolution. The Chebyshev algorithm is a widely known compression technique that employs vector quantization. In [16], the authors applied this technique to compress time series data. This compression applies the Chebyshev transform on data segments, resulting in matrix of Chebyshev coefficients. The selected coefficients are then used to quantize time series segments before being stored on top of a lossless compression. The Chebyshev algorithm performs the same sequence of computations on each data segment. The resulting compression performance depends on the size of the segment, the threshold to restrain specific coefficients, and the number of quantization bits. In [17], the authors introduce an algorithm called SensCompr, which uses Chebyshev for compression to support dynamically selected segment sizes. This method can achieve a compression of data streams by more than 8:1. Chebyshev compression does not however limit the error and may yield significant information loss. We empirically compare CORAD to a Chebyshev-based technique in Section V.

In [18], two piecewise linear approximation (PLA) filtering methods were introduced, *swing filter* and *slide filter*. They represent a time-varying numerical signal by a piecewise linear function, consisting of connected line segments. The *slide filter* method proved to have a better compression ratio than *swing filter*, but it also has a higher space and time complexity. In [19], Luo et al. introduce *mixed-PLA*, an algorithm based on PLA with a guarantee of maximum error. In [20], the authors implement piecewise constant approximation (PCA) with an error bound. They show that PCA performs better than PLA when the time series include many fluctuations and important transitions. Another popular compression approach consists in approximating the data as a sequence of low-order polynomials [17], [21]–[24]. In [21], piecewise polynomial approximation (PPA) was applied for searching in compressed time series data. The authors use a dynamic algorithm that selects the best of piecewise constant, linear, and polynomial algorithm's output for every segment. These approaches focus on specific applications, namely similarity search, and incur a significant error loss to obtain a good space reduction. The paper [25] introduces Spritz, a compression technique that employs integer compression algorithms such as SIMD-BP128 [26], FastPFOR [14], Simple8b [15]. These techniques focus on local compression and thus cannot be applied on large time series streams.

III. Background

This section introduces the main concepts that we will use throughout the paper. We use bold upper-case letters to refer to matrices, regular font upper-case letters to refer to vectors and lower-case letters to refer to elements of vectors/matrices. For example, $\mathbf{X}$ is matrix, X is a set/vector and x_i is the i-th element of X.

A. Definitions

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 timestamps t_i. Time series can be *univariate* (2-dimensional) or *multivariate* (multi-dimensional). In univariate series, a temporal value is a scalar that refers to one specific phenomenon, e.g., temperature. In multivariate series, a value is a vector that refers to multiple phenomena, e.g., temperature, precipitation and humidity.

The *Pearson correlation coefficient (r)* measures the linear relationship between two time series. The absolute value of r ranges between 1 (perfectly correlated) and 0 (not correlated). The Pearson correlation is formally defined as:

$$r = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2}\sqrt{\sum_i (y_i - \bar{y})^2}} \tag{1}$$

where x and y are two time series segments to be compared, and $\bar{x}$ and $\bar{y}$ are their respective means. The result of correlation computation on each window is a correlation matrix $\mathbf{M} \in \mathbb{R}^{n \times n}$ where n is the number of the input time series.

A *dictionary* $\boldsymbol{D}$ is a collection of fixed-length elementary time series segments named atoms. A small number of atoms, combined linearly, can represent most information of any time series segment. A dictionary-based sparse representation (reps. sparse approximation) is a linear combination of dictionary atoms that accounts for all (reps. most) information from time series segments.

B. Dictionary-based sparse coding

The aim of sparse coding methods is to represent time series as a linear combination of few dictionary components (atoms) (Figure 2). These atoms are not necessarily orthogonal. In fact, we can have seemingly redundant dictionary components that allow multiple representations of the same time series segment. This provides an improvement in sparsity and flexibility of the representation. Sparse coding is generally formulated as an optimization problem that minimizes the error of the reconstruction under a sparsity-constraint or under an error-constraint problem.

a) Sparsity-constraint::

$$\hat{\gamma} = \underset{\gamma}{arg\,min} \|x - \mathbf{D}\gamma\|_2^2 \quad \text{s.t.} \quad \|\gamma\|_0 \leq K$$

b) Error-constraint::

$$\hat{\gamma} = \underset{\gamma}{arg\,min} \|\gamma\|_0 \quad \text{s.t.} \quad \|x - \mathbf{D}\gamma\|_2^2 \leq \epsilon$$

The sparsity-constrained approach aims to represent the time series by a linear combination of up to K known atoms. This leads to more compact representations. The error-constrained approach focuses on limiting the squared error of the representation to a certain threshold. This avoids higher representation errors. Both approaches are NP-hard problems [27]. However, efficient techniques such as the matching pursuit method [28] or its orthogonal variant [29], and basis pursuit [30] provide accurate approximations of the optimal solution.

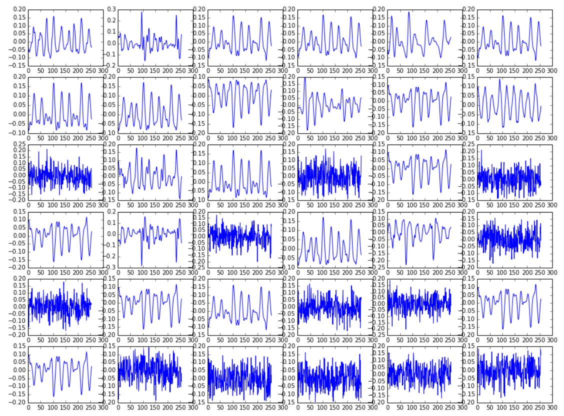


Fig. 2. Example of dictionary atoms of length 256.

IV. Method

In this section, we describe CORAD, our new correlation-aware compression method for time series data streams. CORAD uses a novel sparse dictionary compression techniques leveraging correlation to achieve high compression ratios.

A. Framework

CORAD encodes segments of time series data by using dictionary atoms and inter-time series correlation information. It consists of two main phases: (1) Dictionary Sparse Coding and (2) Compression. In the beginning of CORAD's process, the time series data is normalized, partitioned into segments and the resulting correlation across segments belonging to the same window is computed. In the Dictionary Sparse Coding phase, we identify a small set of dictionary atoms that best approximate each data segment, resulting in a significant reduction in data size. Finally, in the compression phase, the sparse coding and the correlation matrices of each window are combined to achieve an optimal correlation-aware compression. CORAD also allows to bound the maximum error in the compressed data while achieving very high compression ratios.

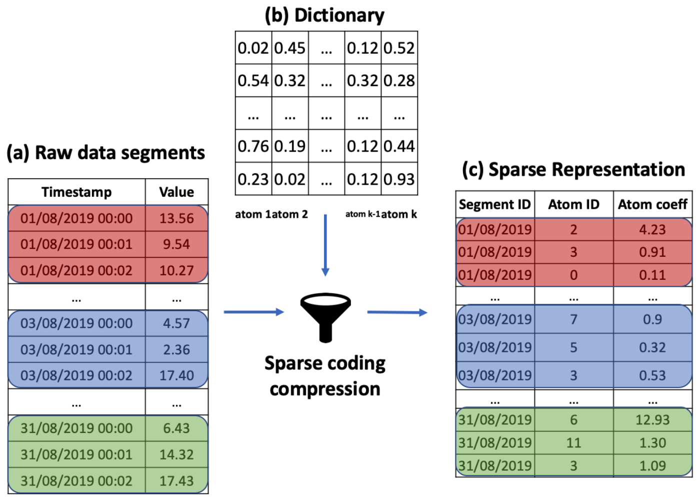
When new data emerges (Figure 3(a)), we check if the new number of data points qualifies to constitute a new data segment (i.e., if they reach a segment's length). Once new data qualifies to form a new segment, the linear correlation across the related time series is computed. The result of the correlation computation on each window is a correlation matrix $M \in \mathbb{R}^{n \times n}$.

B. Sparse dictionary coding

We represent time series segments separately using a small set of the dictionary atoms. The dictionary learning is conducted as an initial configuration step of CORAD (Figure 3(b)). The necessary time for dictionary construction depends on several application parameters (e.g., residual error threshold, number of atoms, etc.). The quality of the dictionary can be evaluated from a compression/querying point of view, the quality of the storage of the compressed representations, and the accuracy of the compressed data. A dictionary can be updated for long time series.

The number of dictionary atoms is an important parameter for its efficiency. A dictionary with redundant atoms allows multiple representations of the same times series and provides potential improvements in terms of the sparsity and flexibility of the representation. When new segments are formed, a dictionary-based sparse representation is determined from the time series segments and stored in a database (Figure 3(c)). The sparse dictionary representation boils down to a set of atoms and their associated coefficients for each time series segment.

Our dictionary sparse coding is derived from Orthogonal Matching Pursuit (OMP) [29], a greedy heuristic to rapidly find a nearly optimal weighted set of a dictionary atoms that approximates each data segment (see Algorithm 1). The atoms are selected incrementally. In each iteration, we select the atom D_j that is most highly correlated to the input sample X or to its residual part r (the current *error*). The coefficient α_j, obtained by an orthogonal projection on the sub-space defined by the atoms selected so far, defines the contribution of D_j to reconstruct X. The process is reiterated until the maximal number ρ of atoms is reached. This phase outputs a weighted list of dictionary atoms (Atom ID, Coefficient).



Timestamp	Value
01/08/2019 00:00	13.56
01/08/2019 00:01	9.54
01/08/2019 00:02	10.27
...	...
03/08/2019 00:00	4.57
03/08/2019 00:01	2.36
03/08/2019 00:02	17.40
...	...
31/08/2019 00:00	6.43
31/08/2019 00:01	14.32
31/08/2019 00:02	17.43

0.02	0.45	...	0.12	0.52
0.54	0.32	...	0.32	0.28
...	...	...	...	...
0.76	0.19	...	0.12	0.44
0.23	0.02	...	0.12	0.93

Segment ID	Atom ID	Atom coeff
01/08/2019	2	4.23
01/08/2019	3	0.91
01/08/2019	0	0.11
...	...	...
03/08/2019	7	0.9
03/08/2019	5	0.32
03/08/2019	3	0.53
...	...	...
31/08/2019	6	12.93
31/08/2019	11	1.30
31/08/2019	3	1.09

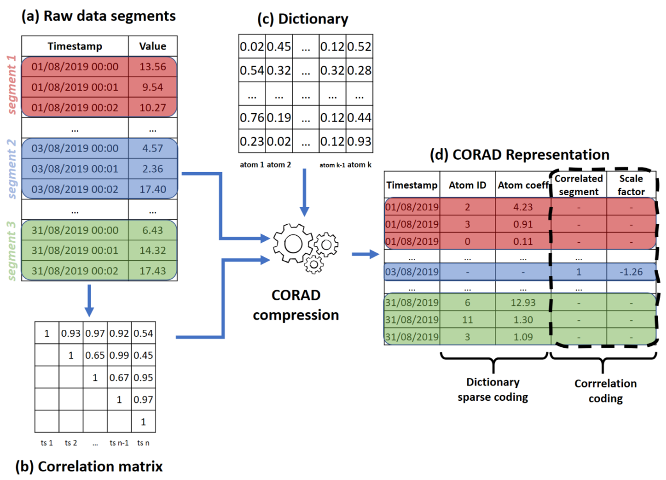
Fig. 3. Schema of dictionary-based sparse coding

Algorithm 1: Dictionary Sparse Coding

```
Input  : X, D, $\rho$
Output: $\Omega, \alpha$
1 r := X ;
2 $\Omega := \{\emptyset\}$;
3 while $|\Omega| \leq \rho$ do
      // select the next atom $D_j$
4     $D_j := \text{argmax}_{j \notin \Omega} \frac{r^T D_j}{\|r\| \|D_j\|}$
      // update the set of selected atoms:
5     $\Omega := \Omega \cup \{j\}$;
      // update the coefficients
6     $\alpha := (D_\Omega^T D_\Omega)^{-1}(D_\Omega^T X)$
      // $D_\Omega$ is the sub-dictionary of the
         selected atoms and $\alpha$ the related
         coefficients;
      // estimate the new residual
7     $r := X - D_\Omega \alpha$;
8 return $\Omega, \alpha$ ;
```

C. Correlation-aware compression

Our final compressed representation uses sparse dictionary coding and correlation information (Figure 4). We sort the correlation matrix computed in the partitioning phase to obtain the most correlated segments first (see Figure 4(b)). We use the absolute sum of the correlation matrix rows as a metric for the weight of a segment's correlation with other segments. The higher the absolute sum of a correlation matrix row, the stronger the correlation of its corresponding segment with other time series. The segments are then stored in reverse order of the sorted correlation matrix rows. We use dictionary representation solely for segments that are not correlated to any other stored segment. We leverage correlation information to store the rest of segments. The process continues until all segments are stored, using either the dictionary atoms or the correlation columns directly (see Figure 4(d)).



(a) Raw data segments

Timestamp	Value
01/08/2019 00:00	13.56
01/08/2019 00:01	9.54
01/08/2019 00:02	10.27
...	...
03/08/2019 00:00	4.57
03/08/2019 00:01	2.36
03/08/2019 00:02	17.40
...	...
31/08/2019 00:00	6.43
31/08/2019 00:01	14.32
31/08/2019 00:02	17.43

(b) Correlation matrix

ts 1	ts 2	...	ts n-1	ts n
1	0.93	0.97	0.92	0.54
	1	0.65	0.99	0.45
		1	0.67	0.95
			1	0.97
				1

(c) Dictionary

atom 1	atom 2	...	atom k-1	atom k
0.02	0.45	...	0.12	0.52
0.54	0.32	...	0.32	0.28
...	...	...	...	...
0.76	0.19	...	0.12	0.44
0.23	0.02	...	0.12	0.93

(d) CORAD Representation

Timestamp	Atom ID	Atom coeff	Correlated segment	Scale factor
01/08/2019	2	4.23	-	-
01/08/2019	3	0.91	-	-
01/08/2019	0	0.11	-	-
...	...	...	...	
03/08/2019	-	-	1	-1.26
...	...	...	...	
31/08/2019	6	12.93	-	-
31/08/2019	11	1.30	-	-
31/08/2019	3	1.09	-	-

Fig. 4. Schema of CORAD compression algorithm

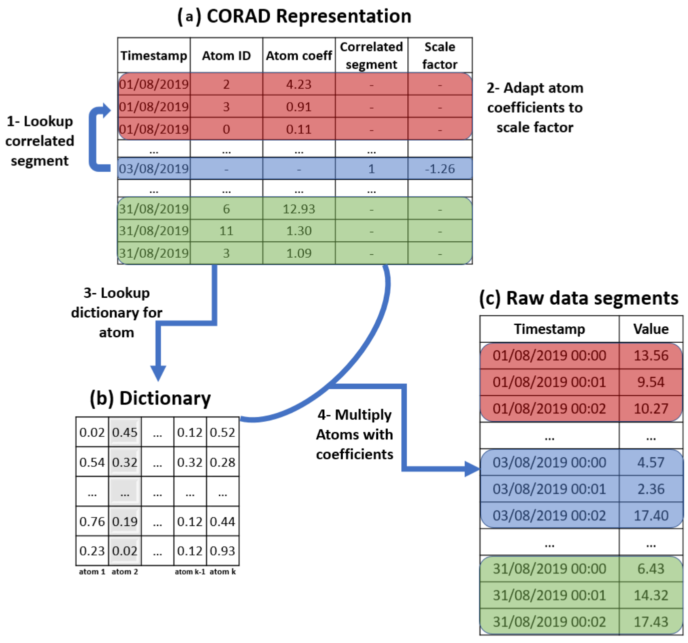
The representation resulting from algorithm 2 limits the usage of dictionary atoms to the case where the data segment is *not* correlated to other segments being inspected. In the example of Figure 4(d), for instance, Segment 2 stores 1 value, instead of 6 values (3 atoms $\times$ 2), reducing the size of stored data.

D. Data reconstruction

CORAD allows to reconstruct the original time series efficiently (Figure 5). In a first step, the dictionary-stored segments are looked up using the correlation-coded segments (see step 1 in Figure 5). Next, the correlation atom coefficients are multiplied by the scale factor to obtain the current segment coefficients (see step 2 in Figure 5). The relevant atoms are next identified and retrieved from the dictionary (see step 3 in Figure 5). Finally, depending on the aggregation level, the corresponding coefficients from the sparse representation are multiplied by the corresponding atoms from the dictionary to reconstruct the original time series (see step 4 in Figure 5).

Algorithm 2: CORAD Compression Pseudo-algorithm

```
Input  : Dictionary D, Window Length $l$, Sparsity level
         (Number of Atoms) $\alpha$, Error threshold $\epsilon_{max}$.
Output: R
// correlation threshold from error
   threshold (section IV-E)
1  $c_{min} := correlation\_threshold(\epsilon_{max})$
2  while stream do
3      $S_{current} := []$;
4      $\widetilde{D}_{stored} := []$;
5      while $length(S_{current}) < l$ do
6          $S_{current} := S_{current} \cup stream$;
       // correlation matrix with rows
          reversely sorted by their sum
7      $\mathbf{C} := sort\_correlation(S_{current}, descending)$;
8      for $t$ in $\mathbf{C}$ do
           // select stored correlated segments
              as candidates
9          $candidates := \{\mathbf{C}_t \mid (\mathbf{C}_{t,c} > c_{min}) \wedge (c \neq t) \wedge (c \in \widetilde{D}_{stored})\}$;
10         if $candidates = \emptyset$ then
               // no already correlated stored
                  segments
11             $\mathbf{R}_{w,t} :=$ Dictionary_Coding(t, D, $\alpha$);
12             $\widetilde{D}_{stored} := \widetilde{D}_{stored} \cup t$;
13         else
14             c $:= \{i \in candidates \mid (\mathbf{C}_i = max(\mathbf{C}_{t,candidates})\}$;
               // store the segment c and its
                  scale factor in the correlation
                  segment
15             $\mathbf{R}_{w,t} :=$ correlation_coding(t, c);
16 return R ;
```


(a) CORAD Representation
1- Lookup correlated segment
2- Adapt atom coefficients to scale factor
3- Lookup dictionary for atom
4- Multiply Atoms with coefficients
(b) Dictionary
(c) Raw data segments

(a) CORAD Representation

Timestamp	Atom ID	Atom coeff	Correlated segment	Scale factor
01/08/2019	2	4.23	-	-
01/08/2019	3	0.91	-	-
01/08/2019	0	0.11	-	-
...	...	...	...	
03/08/2019	-	-	1	-1.26
...	...	...	...	
31/08/2019	6	12.93	-	-
31/08/2019	11	1.30	-	-
31/08/2019	3	1.09	-	-

(b) Dictionary

atom 1	atom 2	...	atom k-1	atom k
0.02	0.45	...	0.12	0.52
0.54	0.32	...	0.32	0.28
...	...	...	...	...
0.76	0.19	...	0.12	0.44
0.23	0.02	...	0.12	0.93

(c) Raw data segments

Timestamp	Value
01/08/2019 00:00	13.56
01/08/2019 00:01	9.54
01/08/2019 00:02	10.27
...	...
03/08/2019 00:00	4.57
03/08/2019 00:01	2.36
03/08/2019 00:02	17.40
...	...
31/08/2019 00:00	6.43
31/08/2019 00:01	14.32
31/08/2019 00:02	17.43

Fig. 5. Overview of query execution over CORAD compressed data

E. Error Thresholding

Dictionary-based compression is lossy, in the sense that the compressed data is (very) similar but not identical to the original data. One highly desirable property of lossy algorithms is the ability to bound the error introduced. In order to do so, one needs to specify not only the number of dictionary atoms to use in sparse coding but also the correlation threshold we use to consider two time series segments as correlated. The correlation threshold is hard to determine beforehand, as it depends on both the application needs and on the dataset at hand.

The error bounding technique we introduce uses the maximum error threshold to determine the minimal correlation threshold and to limit the loss of accuracy. Based on the Pearson correlation coefficient described in Equation (1), and the Mean Squared Error (MSE) equation.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2 \qquad (2)$$

Where x and y are the original and the reconstructed time series segments respectively. We know that different sample means have no influence on the correlation coefficient r. Different means will, however, influence the mean squared error across the segments. To better capture this difference, CORAD uses Z-score normalization in order to bound the data accuracy using the maximum error between the original data and the compressed data. We use the fact that Z-score normalized time series have a zero mean and a standard deviation equal to 1 [31]:

$$\frac{1}{n}\sum_{i=1}^{n}x_i = 0, \quad \frac{1}{n}\sum_{i=1}^{n}y_i = 0, \quad \frac{1}{n}\sum_{i=1}^{n}y_i^2 = 1, \quad \frac{1}{n}\sum_{i=1}^{n}y_i^2 = 1 \qquad (3)$$

From the equations (1) and (3), we can simplify the Pearson correlation coefficient as follows:

$$r = \frac{1}{n}\sum_{i=1}^{n}x_i y_i \qquad (4)$$

From the equations (2), (3), and (4), we can simplify the error measure as follows:

$$\begin{aligned} MSE &= 2\left(1 - \frac{1}{n}\sum_{i=1}^{n}x_i y_i\right) \\ &= 2(1-r) \end{aligned}$$

We obtain a relation between Pearson correlation coefficient (r) and the mean squared error (MSE).

$$MSE = 2(1-r) \qquad (5)$$

CORAD uses the above relation to unify and create a simple relation between the Pearson correlation coefficient and the mean squared error. Z-score normalization is also useful as it removes the shift difference between correlated time series segments. This results in an even more compact representation as the shifts do not have to be stored for each segment. CORAD naturally uses the normalization information to reconstruct the original data.

V. Experimental evaluation

To assess CORAD's performance, we compare it to state-of-the art compression algorithms on a set of publicly available datasets. We vary CORAD's parameters and analyze their effect on the compression performance. The code and raw results are publicly available[1]. All experiments were run on a machine with 2.8GHz quad-core Intel Core i7 processor and 16GB RAM.

A. Datasets

We run our evaluation on several real-world datasets collected from several repositories: the UCR Time Series Classification Archive (UCR) [32], the UCI Machine Learning Repository (UCI) [33], and the Swiss Federal Office for the Environment(FOEN) [2].

The datasets were chosen in order to support different cases in regard to the number of variables and their length. We briefly describe each dataset below.

1) **ACSF1.** This dataset contains 100 time series comprising 1460 data points each. The dataset contains the power consumption of typical appliances. The recordings are characterized by long idle periods and some high bursts of energy consumption when the appliance is active. This dataset was obtained from the UCR respository.
2) **BAFU.** This dataset contains 10 time series comprising 50'000 data points each. It contains hydrological data collected across multiple stations during the period from 1974 to 2015. This dataset was obtained from the FOEN.
3) **GSATM.** This dataset contains 19 time series comprising 4'178'504 data points each. The data originated from gas sensors that were exposed to dynamic mixtures of carbon monoxide (CO) and humid synthetic air in a gas chamber. This dataset was obtained from the UCI respository.
4) **PigAirwayPressure.** This dataset contains 208 time series comprising 2000 data points each. This dataset contains bleeding detection data from vital signs measured at high frequency (250Hz) using a bed-side hemodynamic monitoring system. The data was collected from a cohort of 52 healthy pigs subjected to induced slow bleeding. This dataset was obtained from theUCR respository.
5) **SonyAIBORobotSurface2.** This dataset contains 953 time series comprising 65 data points each. This dataset represents the X-axis of a robot that rolls/pitches/yaws (measured with accelerometers). The associated task

[1]https://github.com/eXascaleInfolab/CORAD
[2]https://www.bafu.admin.ch/bafu/en/home.html

is to detect the surface being walked on (cement or carpet/field). This dataset was obtained from the UCR respository.

6) **Yoga.** This dataset contains 3000 time series comprising 426 data points each. It captures two actors transiting between yoga poses in front of a green screen. Each image was converted to a one dimensional series by identifying an outline and measuring the distance of the outline to the centre. This dataset was obtained from the UCR respository.

Our experiments aim at illustrating the performance of CORAD in terms of: (1) compression ratio, (2) accuracy, (3) runtime, and (4) usability, where usability refers to the ability of a compression algorithm to produce usable compressed data for an application. We use 3 datasets to compare amongst algorithms of different sizes: medium (ACSF1), large (BAFU) and very large (GSATM). We also run tests on 5 datasets of different sizes (ACSF1, BAFU, PigAirway, Sony, Yoga) varying parameters of the algorithm.

B. Compression ratio

We start by studying the compression ratio of CORAD. We measure the compression efficiency using the ratio between the original data size and the compressed size.

$$\text{Compression ratio} = \frac{\text{Original Data Size}}{\text{Compressed Data Size}}$$

We first study CORAD's performance by running tests and compare its performance with three baselines techniques, TRISTAN (our former sparse compression technique described in [2]), DWT (a compression technique using wavelets to decompose time series into frequency components) and Chebyshev (a widely used piecewise compression technique that is vector quantization based). The window length selected for this experiment is 20 values, with a maximum MSE error bound equal to 0.4 and a sparsity level (number of dictionary atoms used for sparse coding) equal to 4 atoms. Figure 6 illustrates the compression ratio results of CORAD along with the baseline techniques. CORAD outperforms all the baseline techniques.

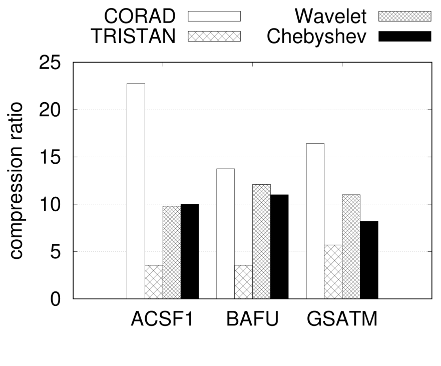


Fig. 6. Compression ratios.

We now vary the parameters of CORAD and observe their effect on the compression ratio. We first vary the maximum error threshold. We use a 20 values segment length and 4 dictionary atoms for the sparse coding. Figure 7(a) illustrates how the compression ratio varies with the maximum error threshold. We observe that the larger the maximum error threshold, the higher compression ratios CORAD yields. This is due to the increasing usage of correlation encoding as opposed to dictionary sparse coding, which results in a better utilization of storage.

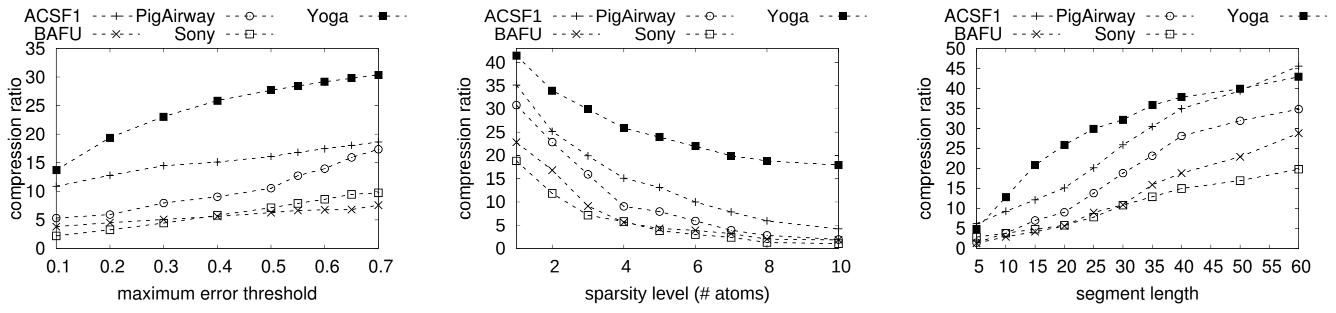
We now vary the sparsity level. We use a 20 values segment length and a maximum MSE error bound of 0.4. Figure 7(b) illustrates how the compression ratio varies with the sparsity level. When the number of atoms used to represent one time series segment increases, it results in more information stored in the sparse coding, and the compression shall be negatively impacted.

Next, we analyze how CORAD's compression ratio behaves when the window length varies. We vary the window length while using a maximum MSE error bound equal to 0.4 and a sparsity level equal to 4 atoms. Figure 7(c) shows that the compression ratio increases when the segments are longer. This is because our segmented windows do not overlap, and an increase in the size of a window thus results in having a lower number of windows. This reduction in the number of windows lowers the amount of data stored and therefore the compression gain increases. This could, however, result in a larger loss in accuracy, which highlights the need for our error bounding method.

C. Accuracy

We turn to the study of the loss in information caused by the compression. In order to quantify the accuracy of a reconstruction, we use the Mean Squared Error (MSE) which is the sum of the squares of the differences between the original and the reconstructed values. We analyse the Mean Squared Error (MSE) achieved by each baseline compression algorithms TRISTAN, DWT and Chebyshev, along with that of CORAD. The segment length used is 20 while the maximum error threshold is 0.4 and the sparsity level corresponds to 4 atoms. Figure 8 demonstrates that our previous method, TRISTAN, introduces slightly less error than CORAD, especially on the GSATM Dataset. This slight increase in error is expected due to the weak correlation in this dataset and its relatively low number of time series. We can notice the effect of our error thresholding method, in fact, none of the 3 datasets results in a larger error than the input maximum error thresold (Maximum Error Threshold = 0.4). CORAD outperforms the Chebychev algorithm for all datasets and outperforms the DWT approach in both the BAFU and ACSF1 Datasets. The Wavelet-based technique yields however a more precise representation on the GSATM dataset, due to its weak correlation.

Next, we evaluate the optimal number of atoms to use for a one segment representation. More precisely, we compute the minimum number of atoms to use to approximate the segment properly. The segment length used is 20, with a maximum MSE error threshold equal to 0.4 and a sparsity level equal to 4 atoms. Figure 9 shows that for the three datasets,



(a) Varying Maximum Error. (b) Varying Sparsity Level. (c) Varying Window length.

Fig. 7. CORAD Compression Ratio.

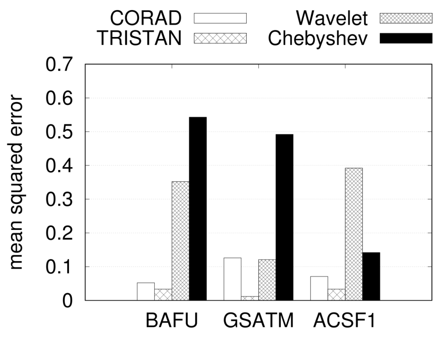


Fig. 8. Accuracy Loss of Different Algorithms.

increasing the number of dictionary atoms to use for a one segment representation does not have an observable impact when $[3-6] <$ sparsity level. This result points us to the minimum number of atoms one can use to obtain a reasonable representation of the original data.

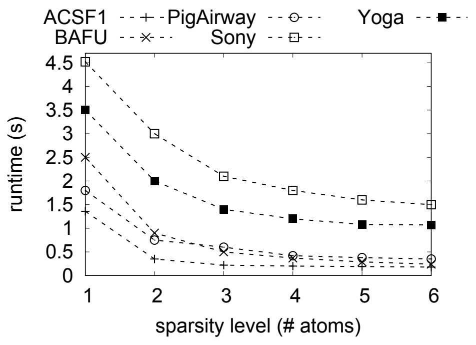


Fig. 9. Accuracy loss vs. Sparsity Level.

D. Runtime

We now turn to the time necessary for compression and decompression. Even though CORAD is capable of executing queries on the compressed data directly (similarly as in [2]) extremely efficiently, we aim to measure the time needed for both compressing and decompressing the data and compare it to existing baselines. The runtimes reported below are averaged over ten consecutive runs.

We consider the runtime of compression and reconstruction achieved by each baseline compression algorithm: TRISTAN, DWT and Chebyshev, and we compare them to that of CORAD. We run our tests on 3 datasets. The segment length used is 20, with a maximum error threshold equal to 0.4 and a sparsity level equal to 4 atoms. Figure 10 shows that CORAD's runtime is faster than DWT and Chebychev approaches. CORAD computes the correlation matrix in addition to TRISTAN method and requires two lookups instead of one when reconstructing the data. This implies, naturally, that TRISTAN's runtime would slightly outperform it.

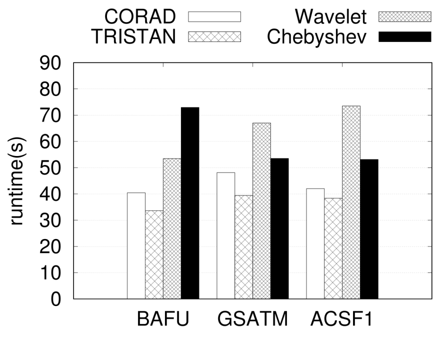
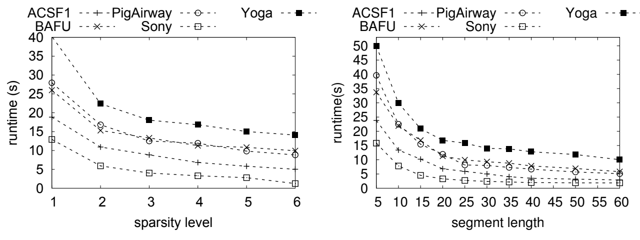


Fig. 10. Runtime of Different Algorithms.

We now evaluate the effect of the sparsity level and of the window length on the compression and reconstruction runtime. Figure 11(a) illustrates that the runtime is proportional to the number of atoms. This is due to the fact that having more atoms for one segment would imply more computation for the compression and decompression. This is yet another reason to use a low number of atoms. On the other hand, increasing the time series segment length results in a lower number of windows and therefore fewer computations for both compression and decompression as Figure 11(b) illustrates.

E. Analytics on compressed data

To evaluate the end-to-end accuracy of our compression scheme in practice, we consider the sample task of recovering missing value from time series. We incrementally delete 20% of the original data, which we recover using some state-of-the-art missing values recovery techniques. We use TKCM [34]



(a) Varying Sparsity Level. (b) Varying Window Length.

Fig. 11. CORAD Efficiency.

and TRMF [35] as recovery methods on the BAFU dataset on both the raw and the decompressed data. Table I shows that the precision results from both versions are very similar to each other. This results validate the assumption that our reconstruction is almost identical to the original data, and that CORAD can be highly effective in a practical setting. As a result, one can use the highly effective representation resulting from our technique without introducing any noticeable artifacts in real-world applications.

TABLE I
Recovery Accuracy on Real-word Datasets

		varying % of miss. val.			
	Method	20 %	40 %	60 %	80 %
Original	TRMF	0.246	0.379	0.363	0.338
	TKCM	1.283	1.421	1.261	1.248
Compr.	TRMF	0.246	0.379	0.363	0.338
	TKCM	1.283	1.421	1.261	1.248

VI. Conclusion

We introduced CORAD, a new real-time technique to effectively compress time series streams. CORAD relies on a dictionary-based technique that exploits the correlation across time series. In addition, CORAD allows to adjust the degree of accuracy that is acceptable depending on the use-case.

We extensively evaluated our method on several real-world datasets and showed that it drastically increases the compression ratios (up to 40:1) compared to state-of-the-art techniques with no significant information loss. As future work, we plan to combine other compression approaches with CORAD such as scalar quantization compression schemes. As scalar quantization is local, it can be run on top of CORAD to further increase the compression ratio. We also plan to identify the types of queries that are the most efficiently executed on our compressed representation directly.

Acknowledgement

This work was supported by the SNSF project No. 169840 (ProvDS: Uncertain Provenance Management over Incomplete Linked Data Streams).

References

[1] D. Giouroukis, J. Hülsmann, J. von Bleichert, M. Geldenhuys, T. Stullich, F. O. Gutierrez, J. Traub, K. Beedkar, and V. Markl, "Resense: Transparent record and replay of sensor data in the internet of things," in *EDBT*, 2019.

[2] A. Marascu, P. Pompey, E. Bouillet, M. Wurst, O. Verscheure, M. Grund, and P. Cudre-Mauroux, "Tristan: Real-time analytics on massive time series using sparse dictionary compression," in *2014 IEEE International Conference on Big Data (Big Data)*, Oct 2014, pp. 291–300.

[3] Q. Wang and V. Megalooikonomou, "A dimensionality reduction technique for efficient time series similarity analysis," *Information systems*, vol. 33, no. 1, pp. 115–132, 2008.

[4] W. Lang, M. Morse, and J. Patel, "Dictionary-based compression for long time-series similarity," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 22, pp. 1609 – 1622, 12 2010.

[5] B. Hu, Y. Chen, and E. J. Keogh, "Time series classification under more realistic assumptions," in *SDM*, 2013.

[6] P. Chaovalit, A. Gangopadhyay, G. Karabatis, and Z. Chen, "Discrete wavelet transform-based time series analysis and mining," *ACM Computing Surveys (CSUR)*, vol. 43, no. 2, p. 6, 2011.

[7] M. L. Hilton, "Wavelet and wavelet packet compression of electrocardiograms," *IEEE Transactions on Biomedical Engineering*, vol. 44, no. 5, pp. 394–402, 1997.

[8] B. Walczak and D. Massart, "Noise suppression and signal compression using the wavelet packet transform," *Chemometrics and Intelligent Laboratory Systems*, vol. 36, no. 2, pp. 81 – 94, 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0169743996000779

[9] J. L. Cárdenas-Barrera, J. V. Lorenzo-Ginori, and E. Rodríguez-Valdivia, "A wavelet-packets based algorithm for eeg signal compression," *Medical Informatics and the Internet in Medicine*, vol. 29, no. 1, pp. 15–27, 2004. [Online]. Available: https://doi.org/10.1080/14639230310001636499

[10] F. C. A. Fernandes, R. L. C. van Spaendonck, and C. S. Burrus, "Multidimensional, mapping-based complex wavelet transforms," *IEEE Transactions on Image Processing*, vol. 14, no. 1, pp. 110–124, Jan 2005.

[11] T. M. Martinetz, S. G. Berkovich, and K. J. Schulten, "'neural-gas' network for vector quantization and its application to time-series prediction," *IEEE transactions on neural networks*, vol. 4, no. 4, pp. 558–569, 1993.

[12] A. Gersho and R. M. Gray, *Vector quantization and signal compression*. Springer Science & Business Media, 2012, vol. 159.

[13] J. Deboever, S. Grijalva, M. J. Reno, and R. J. Broderick, "Fast quasi-static time-series (qsts) for yearlong pv impact studies using vector quantization," *Solar Energy*, vol. 159, pp. 538–547, 2018.

[14] D. Daniel and L. Boytsov, "Decoding billions of integers per second through vectorization," *Software: Practice and Experience*, vol. 45, no. 1, pp. 1–29, 2015.

[15] V. N. Anh and A. Moffat, "Index compression using 64-bit words," *Softw. Pract. Exper.*, vol. 40, no. 2, pp. 131–147, Feb. 2010. [Online]. Available: http://dx.doi.org/10.1002/spe.v40:2

[16] S. E. Hawkins and E. H. Darlington, "Algorithm for compressing time series data," *Journal of Machine Learning Research*, vol. 11, no. Jan, pp. 19–60, 2012. [Online]. Available: http://ntrs.nasa.gov/search.jsp?R=20120010460

[17] A. Ukil, S. Bandyopadhyay, and A. Pal, "Iot data compression: Sensor-agnostic approach," in *2015 Data Compression Conference*, April 2015, pp. 303–312.

[18] H. Elmeleegy, A. Elmagarmid, E. Cecchet, W. Aref, and W. Zwaenepoel, "Online piece-wise linear approximation of numerical streams with precision guarantees," *Proceedings of the VLDB Endowment*, vol. 2, 08 2009.

[19] G. Luo, K. Yi, S. Cheng, Z. Li, W. Fan, C. He, and Y. Mu, "Piecewise linear approximation of streaming time series data with max-error guarantees," in *2015 IEEE 31st International Conference on Data Engineering*, April 2015, pp. 173–184.

[20] R. Goldstein, M. Glueck, and A. Khan, "Real-time compression of time series building performance data," in *Proceedings of IBPSA-AIRAH Building Simulation Conference*, 2011.

[21] F. Eichinger, P. Efros, S. Karnouskos, and K. Böhm, "A time-series compression technique and its application to the smart grid," *The VLDB Journal*, vol. 24, no. 2, pp. 193–218, Apr. 2015.

[22] E. Keogh, S. Chu, D. Hart, and M. Pazzani, "An online algorithm for segmenting time series," in *Proceedings 2001 IEEE International Conference on Data Mining*, Nov 2001, pp. 289–296.

[23] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra, “Dimensionality reduction for fast similarity search in large time series databases,” *Knowledge and Information Systems*, vol. 3, no. 3, pp. 263–286, Aug 2001. [Online]. Available: https://doi.org/10.1007/PL00011669

[24] D. Lemire, “A better alternative to piecewise linear time series segmentation,” *CoRR*, vol. abs/cs/0605103, 2006. [Online]. Available: http://arxiv.org/abs/cs/0605103

[25] D. W. Blalock, S. Madden, and J. V. Guttag, “Sprintz: Time series compression for the internet of things,” *IMWUT*, vol. 2, pp. 93:1–93:23, 2018.

[26] W. X. Zhao, X. Zhang, D. Lemire, D. Shan, J. Nie, H. Yan, and J. Wen, “A general simd-based approach to accelerating compression algorithms,” *CoRR*, vol. abs/1502.01916, 2015. [Online]. Available: http://arxiv.org/abs/1502.01916

[27] M. E. Davies and T. Blumensath, “Faster greedier: algorithms for sparse reconstruction of large datasets,” in *2008 3rd International Symposium on Communications, Control and Signal Processing*, March 2008, pp. 774–779.

[28] S. G. Mallat and Z. Zhang, “Matching pursuits with time-frequency dictionaries,” *IEEE Transactions on signal processing*, vol. 41, no. 12, pp. 3397–3415, 1993.

[29] Y. C. Pati, R. Rezaiifar, and P. S. Krishnaprasad, “Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition,” in *Proceedings of 27th Asilomar conference on signals, systems and computers*. IEEE, 1993, pp. 40–44.

[30] S. Chen, D. Donoho, and M. Saunders, “Atomic decomposition by basis pursuit,” *SIAM Review*, vol. 43, no. 1, pp. 129–159, 2001. [Online]. Available: https://doi.org/10.1137/S003614450037906X

[31] M. D. Pettersen, W. Du, M. E. Skeens, and R. A. Humes, “Regression equations for calculation of z scores of cardiac structures in a large cohort of healthy infants, children, and adolescents: An echocardiographic study,” *Journal of the American Society of Echocardiography*, vol. 21, no. 8, pp. 922 – 934, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0894731708001193

[32] H. A. Dau, E. Keogh, K. Kamgar, C.-C. M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratanamahatana, Yanping, B. Hu, N. Begum, A. Bagnall, A. Mueen, and G. Batista, “The ucr time series classification archive,” October 2018.

[33] D. Dua and C. Graff, “UCI machine learning repository,” 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[34] K. Wellenzohn, M. H. Böhlen, A. Dignös, J. Gamper, and H. Mitterer, “Continuous imputation of missing values in streams of pattern-determining time series,” 2017.

[35] H.-F. Yu, N. Rao, and I. S. Dhillon, “Temporal regularized matrix factorization for high-dimensional time series prediction,” in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 847–855. [Online]. Available: http://papers.nips.cc/paper/6160-temporal-regularized-matrix-factorization-for-high-dimensional-time-series-prediction.pdf