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 they 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.

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.
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, \( X \) 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:

\[
r = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2}\sqrt{\sum_i (y_i - \bar{y})^2}}
\]

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 \( M \in \mathbb{R}^{n \times n} \) where \( n \) is the number of the input time series.

A dictionary \( D \) 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.

1) Sparsity-constraint:

\[
\hat{\gamma} = \arg\min_{\gamma} \|x - D\gamma\|_2^2 \quad \text{s.t.} \quad \|\gamma\|_0 \leq K
\]

2) Error-constraint:

\[
\hat{\gamma} = \arg\min_{\gamma} \|\gamma\|_0 \quad \text{s.t.} \quad \|x - D\gamma\|_2^2 \leq \epsilon
\]

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 $\alpha_j$, 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 $\rho$ of atoms is reached. This phase outputs a weighted list of dictionary atoms (Atom ID, Coefficient).

Algorithm 1: Dictionary Sparse Coding

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

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)).

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 × 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 \( \epsilon_{\text{max}} \).

**Output:** \( R \)

// correlation threshold from error threshold (section IV-E)

1. \( c_{\text{min}} \leftarrow \text{correlation\_threshold}(\epsilon_{\text{max}}) \)

2. while stream do
   3. \( S_{\text{current}} \leftarrow []; \)
   4. \( D_{\text{stored}} \leftarrow []; \)
   5. while \( \text{length}(S_{\text{current}}) < l \) do
      6. \( S_{\text{current}} \leftarrow S_{\text{current}} \cup \text{stream}; \)
      7. \( C \leftarrow \text{sort\_correlation}(S_{\text{current}}, \text{descending}); \)
      8. for \( t \) in \( C \) do
         9. \( \text{candidates} \leftarrow \{ C_t | (C_{t,c} > c_{\text{min}}) \wedge (c \neq t) \wedge (c \in D_{\text{stored}}) \}; \)
         10. if \( \text{candidates} = \emptyset \) then
              11. \( R_{w,t} \leftarrow \text{Dictionary\_Coding}(t, D, \alpha); \)
              12. \( D_{\text{stored}} \leftarrow D_{\text{stored}} \cup t; \)
         else
            13. \( c \leftarrow \{ i \in \text{candidates} | (C_i = \max(C_{t,\text{candidates}})) \}; \)
            14. \( R_{w,t} \leftarrow \text{correlation\_coding}(t, c); \)
   15. return \( R \);

---

**Fig. 4. Schema of CORAD compression algorithm**

**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.

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

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]:

\[ \frac{1}{n} \sum_{i=1}^{n} x_i = 0, \quad \frac{1}{n} \sum_{i=1}^{n} y_i = 0, \quad \frac{1}{n} \sum_{i=1}^{n} x_i^2 = 1, \quad \frac{1}{n} \sum_{i=1}^{n} y_i^2 = 1 \]  

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

\[ r = \frac{1}{n} \sum_{i=1}^{n} x_i y_i \]  

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

\[ MSE = 2 \left( 1 - \frac{1}{n} \sum_{i=1}^{n} x_i y_i \right) \]

\[ = 2(1 - r) \]

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

\[ MSE = 2(1 - r) \]  

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 repository.

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 repository.

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 the UCR repository.

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 repository.

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 repository.

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.](image)

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 threshold (Maximum Error Threshold = 0.4). CORAD outperforms the Chebyshev 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,
increasing the number of dictionary atoms to use for a one segment representation does not have an observable impact when \(3 < \text{sparsity level} < 6\). This result points us to the minimum number of atoms one can use to obtain a reasonable representation of the original data.

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.

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.

### 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.

### 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]
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.

### 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


