# TRISTAN: Real-Time Analytics on Massive Time Series Using Sparse Dictionary Compression

Alice Marascu, Pascal Pompey, Eric Bouillet,
Michael Wurst, Olivier Verscheure
IBM Research

Martin Grund, Philippe Cudre-Mauroux
eXascale Infolab
University of Fribourg—Switzerland

***Abstract*—Large-scale critical infrastructures such as transportation, energy, or water distribution networks are increasingly equipped with smart sensor technologies. Low-latency analytics on the resulting times series would open the door to many exciting opportunities to improve our grasp on complex urban systems. However, sensor-generated time series often turn out to be noisy, non-uniformly sampled, and misaligned in practice, making them ill-suited for traditional data processing. In this paper, we introduce TRISTAN (massive TRIckletS Time series ANalysis), a new data management system for efficient storage and real-time processing of fine-grained time series data. TRISTAN relies on a dedicated, compressed sparse representation of the time series using a dictionary. In contrast to previous approaches, TRISTAN is able to execute most analytics queries on the compressed data directly, and supports efficient and approximate query answering based on the most significant atoms of the dictionary only. We present the overall architecture of our system and discuss its performance on several smarter city datasets, showing that TRISTAN can achieve up to 20:1 compression ratios and 250x speedup compared to a state-of-the-art system.**

***Keywords*-tricklets; dictionary compression; matching pursuit;**

## I. Introduction

Providers of large-scale critical infrastructure such as intelligent transportation networks, building monitoring systems, energy grids, and water distribution networks are setting increasingly high expectations in terms of continuous, low-latency data analysis (i.e., answer given within a few seconds) and monitoring of their infrastructures. Many providers have adopted smart sensor technologies to assist them in their operations. In the energy domain, for instance, the widespread installation of smart meters and aggressive policies for renewable energy production and energy conservation call for the monitoring and management of millions of sensor-generated time series to correctly analyze the energy load at various granularities throughout the electrical grid.

Sensor-instrumented infrastructures are offering an affordable and scalable way of collecting Big Data, resulting in a treasure trove of fine-grained information. The finer granularity of the node deployment does not necessarily imply a continuous spatiotemporal space in terms of resulting information, however, but rather a large population of sensors that produce many redundant and misaligned time series. We refer to such time series as *tricklets* and focus our attention on this type of data in this paper. Monitoring tricklets opens the door to new opportunities in data mining and analytics, but also to new challenges in terms of data storage and processing.

The vast majority of existing tricklets processing methods use some sort of smoothing to remove some of the irregularities of the underlying sensor data. Many approaches for instance expect the time series to be uniformly sampled, temporally aligned, with no or only a few data points missing. Applying analytics on such data automatically leads to distorted results. In reality, the sensor may experience a failure or may deliberately not transmit measurements in order to preserve its battery life, resulting in non-uniform sampling rates. Temporal misalignment between several time series may occur due to a variety of causes (e.g., unsynchronized sensor clocks, hardware faults, or network delays). Such data misalignments introduce additional errors in the latter stages of data analysis, since domain information must be considered in order to decide on the correct alignment. In addition, as sensors streams are typically sent in discrete form, analytics tools might in turn transform the samples back into a continuous domain, thereby introducing additional discretization errors.

Real-word sensor data are also in our experience surprisingly noisy. Learning how to distinguish good data samples from noise is a process that is hard to generalize to several contexts, and that requires a complex analysis of both application-specific factors and external factors. For instance, a data point may be considered as an outlier in one domain of application but not in another because of subtle differences in the domain's respective physical properties. In a similar fashion, background domain knowledge can often be used to infer missing data. Hence, it is important to apply analytics on a faithful representation of the time series that is as close as possible to the raw data. Data sampling rate plays an important role in that context; Obviously, a higher sampling rate avoids temporal aliasing, improves resolution and reduces noise, which leads to less uncertainty in the latter processing steps and to the extraction of more domain information. However, a higher sampling rate also demands higher processing and storage capabilities.

Querying large amounts of time series is a popular research topic and several systems have been proposed in the past few years to tackle this problem (see for instance [22], [1], [30]). However, such systems typically do not offer a specialized storage solution for compactly and faithfully storing the entire stream history, and for efficiently running interactive queries

on top of it; They rather focus on executing queries on recent sample windows, or on optimizing specific types of queries such as similarity queries.

Running interactive analytics on top of very large sets of tricklets data is a challenging task. As the amount of time series data increases, however, it becomes more effective to store compressed versions of tricklets. At query execution time, two options are then available: decompressing the data first or processing the compressed data directly. The second option typically provides far better execution times, but also requires new operators and query plans. Several systems exist that perform computation on compressed time series (see our related work section below) by using Fourier transforms or polynomial approximations. However, these systems only provide a very limited support for spikes analyses or for correlation queries on the noisy, misaligned, and non-uniformly sampled tricklets described above.

Motivated by the above observations and by our own real-world experience analyzing many large-scale Smarter Cities datasets, we decided to design and implement a system specialized in the long-term archival and interactive querying of massive amounts of tricklets. In this paper, we introduce TRISTAN (massive TRIckletS Time series ANalysis), a new database system compactly storing very large amounts of tricklets and supporting real-time analytics on the compressed data directly. Our system includes a sparse dictionary-based time series representation that encodes domain information effectively using matching-pursuit techniques, and implements specialized operators to support efficient analytics queries on top of it. We show that our dictionary-based representation is particularly well-suited to smart cities sensor data and achieves up to 20:1 compression ratios and 250x speedup for query execution compared to state-of-the-art techniques.

The rest of the paper is organized as follows: Section II introduces a few definitions that will be used throughout the paper. The detailed description of our system and some of its extensions are presented in Sections III and IV respectively. Experimental results on two real data sets are presented in Section V. We give an overview of related work in Section VI, and discuss their advantages and drawbacks with respect to TRISTAN. Finally, we conclude our paper in Section VII.

## II. Problem Definition and Notations

In this section, we introduce our problem as well as the terms we use throughout the rest of this paper.

**Definition:** A *time series* is an ordered sequence of $n$ pairs of real value numbers $T = \{(t_1, v_1), ..., (t_n, v_n)\}$, where $t_i$ is the time when value $v_i \in \mathbb{R}$ was registered, with $1 \leq i \leq n$. Timestamp $t_1$ corresponds to the oldest registered value, and timestamp $t_n$ corresponds to the most recently registered value. When ignoring the timestamps (e.g., for uniformly sampled time series), we can write a time series as $T = \{v_1, ..., v_n\}$, with $v_i \in \mathbb{R}$, $1 \leq i \leq n$, or, in a simplified way, $T \in \mathbb{R}^n$.

**Definition:** A *tricklet* is a time series which exhibits some irregularities, i.e., a time series that is non-uniformly sampled, noisy, and/or misaligned. Example of tricklets are time series generated by sensors that monitor a continuous activity. Sensor data are directly impacted by human's seasonal (e.g., daily) behavior, and the tricklets inherit from this trait and present *recurring patterns*, but that are hard to be noticed under the raw and apparently irregular format of the time series.

**Definition:** We call a *tricklet segment* the part of the tricklet contained within a fixed-length time interval aligned along the tricklet's seasonality (e.g., daily tricklet segment). A tricklet consists of a (potentially infinite) sequence of tricklet segments of equal time intervals. A tricklet segment is identified by the beginning of its time interval. Because tricklets can be non-uniformly sampled, it is possible that two tricklet segments contain a different number of values.

**Definition:** A *dictionary* is a collection of fixed-length elementary time series called *atoms*. The atoms are chosen such that most or all information of any tricklet segment can be represented with a linear combination of a small number of atoms from the dictionary.

**Definition:** A *sparse representation* (reps. *sparse approximation*) is a linear combination of atoms from a dictionary that accounts for all (reps. most) information from a tricklet segment.

**Definition:** $\mathcal{Q}$ is a workload defined as a (weighted) list of *analytics queries*, including correlations, sum, and average queries.

**Problem definition:** Given $\mathcal{W}$, a very large stream warehouse[11] of many tricklets, our problem is to design a system able to: 1) effectively segment and compresses $\mathcal{W}$ data by integrating domain information, 2) compactly store compressed data as well as other, third-party data in a database, 3) efficiently execute analytics queries, whenever possible on the compressed representation directly, and 4) trade accuracy levels (with bounded errors) for storage and query execution efficiency.

## III. System Description

This section contains the description of TRISTAN, our proposed solution for the problem introduced above. TRISTAN applies sparse dictionary compression without any restriction on the time series at hand, but is especially efficient on tricklets, which is the focus of this paper. We originally presented the vision behind TRISTAN in [19]. Below, we describe the implementation of this vision. We start by giving an overview of our system below, before delving into query execution and the system's extensions.

### A. Overview of TRISTAN

The main intuition behind our system is to explicitly store and track only the most relevant (i.e., informative) parts of the tricklets in a new optimized representation, and to perform queries directly on top of this optimized representation without the need to decompress the signal.

Figures 1 and 2 give an illustration of our technique, starting with the very first step when the tricklets are cut into batches of tricklet segments of a specific time interval (*Tricklet segments*

in Figure 1). In the case of streaming tricklets, a tricklet segment is considered complete and ready to be compressed when the next timestamp falls outside the time-interval of that tricklet segment. Figure 2 illustrates the processing of a tricklet segment. This process is executed whenever a new tricklet segment is ready for compression, which depends on the seasonality of the tricklet (e.g., daily). Before the first trickle segment can be compressed, a dictionary must be acquired in a preprocessing step (top of Figure 2: *Dictionary*). This operation must be conducted as an initial configuration step of TRISTAN, and more rarely when new patterns emerge in the tricklets that cannot be approximated using the existing dictionary. In the encoding step (from left to right in Figure 2), dictionary-based sparse representations are determined from the tricklet segments and stored in a database. As a last step, analytics queries are run directly on the dictionary-based sparse representations from the database. We describe the above steps in further details in the next sections.

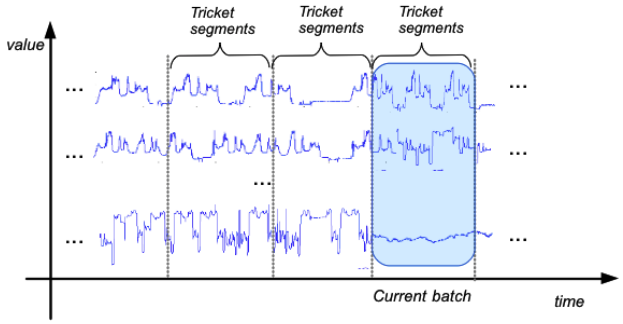


Fig. 1. TRISTAN: Batch processing

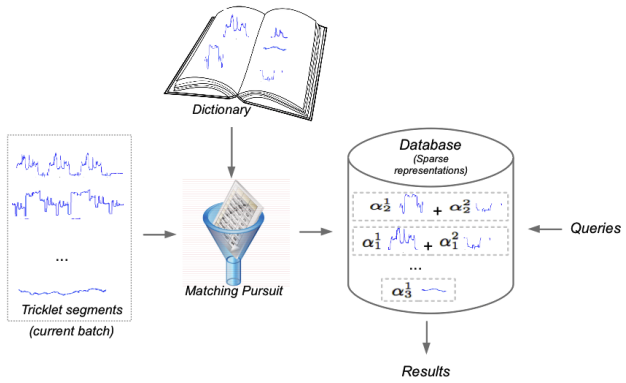


Fig. 2. TRISTAN: overview

### B. Sparse Dictionary Encoding

The tricklet segments are encoded by projecting them on a few atoms of the dictionary, hence obtaining a representation of the segments as a *weighted sum* of several of the dictionary atoms. For this purpose, two subproblems need to be tackled: 1) project the tricklet segment on the atoms of this dictionary, along with 2) finding the dictionary.

Let $T_B = \{v_1, \ldots, v_n\}$ be a tricklet segment of size $n$, $n \in \mathbb{N}^*$(we initially consider uniform time sampling, and for the sake of simplicity ignore the temporal stamp associated to each value). Let $D = \left[ \alpha_1, \ \ldots, \ \alpha_k \right]$ be a dictionary with $k$ *atoms* of $q$ values each. We do not impose any restriction on the dictionary. In particular, the dictionary can be overcomplete, i.e., the size of the dictionary might exceed the dimension of the tricklet space such that any tricklet segment can be represented by more than one combination of atoms.

A sparse representation $S$ on $D$ with a sparsity $sp \leq k$ is:

$$S = \sum_{i=1}^{k} w_i \alpha_i \tag{1}$$

where only $sp$ of the $k$ coefficients $w_i$ are non-zero. In vector notation, this becomes $S = wD$, with $w = [w_1 \ldots w_k]^T$.

Solving point 1) translates into solving Equation 1 for $w$ with $T_B = S$. Given a dictionary, computing the best sparse representation of a tricklet segment is an *NP-hard* problem (as is the problem of encoding a signal given a dictionary) with a reduction to NP-complete problems of subsets selection. One way of computing an approximation of its solution is by using *matching pursuit methods*. Encoding the tricklet segment $T_B$ over the dictionary $D$ leads to solving the following optimization problem:

$$\begin{aligned} \varphi_{sp} &= arg\min_{\mathbf{w}} \|\mathbf{w}D - T_B\|_2 \\ \text{under constraint} &\quad \|\varphi_{sp}\|_0 \leq sp \end{aligned} \tag{2}$$

where $\|.\|_2$ is the $L_2$ norm (sum of squared coefficients) and $\|.\|_0$ is the $L_0$ norm (numbers of non zero coefficients).

One valuable characteristic of this representation is that the decreasing order of the coefficients is also the decreasing order of their weights in the representation, i.e., the first coefficient has a higher participation in the representation than the second one, the second one than the third one, etc. (see subsection III-D for more detail).

Concerning step 2), the *dictionary computation*, TRISTAN can use any dictionary that can be provided either as external knowledge (e.g., by domain experts who know the typical patterns occurring in the time series or who are of particular interest), or learnt, or even a combination of external knowledge and learning. Learning the dictionary is done by using a training set of numerous signals $\{T_{B_1}, \ldots, T_{B_M}\}$ and searching for a dictionary minimizing

$$\begin{aligned} D^* &= arg\min_{D} \sum_{i=1}^{M} \|\varphi_i D - T_{B_i}\|_2 \\ \text{under constraint} &\quad \|\varphi_i\|_0 \leq sp \end{aligned} \tag{3}$$

Clearly, the more trained the dictionary and the more signals get involved in the training process, the better the quality of the obtained dictionary. Solving Equation (3) is NP-hard, but this problem was extensively studied in the literature due to its multiple applications, and there exist several algorithms yielding reasonable performance in practice (among them: [17], [29], [23], [15], [27], [6], [4]).

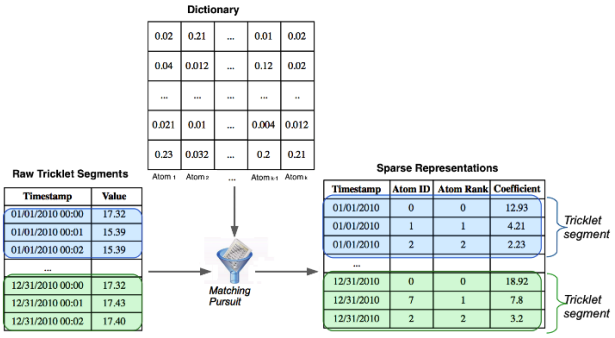


Fig. 3. Schema of TRISTAN database implementation

### C. Database Implementation

Once the dictionary and the compressed representation are created as described above, we can start querying the data. Since the original raw data can contain millions of data points, the goal must be to efficiently store the data and to perform query execution directly on the compressed data without decompressing the complete dataset each time.

We customized the storage layer of the HYRISE hybrid database system [12] in order to realize the full potential of our approach, which is discussed below. We are also currently working on extending DB2 with the required functionality to take full advantage of the sparse representation. Modifying a given database system to integrate the present approach is relatively straightforward, though it requires the implementation of a new operator as we discuss below. However, the resulting performance may vary depending on the system chosen (in that sense, HYRISE is a sensible choice, since it is efficient both for transactional operations and for analytics.)

The dictionary that is used to encode the data is basically a dense matrix containing as many columns as distinct atoms and as many rows as data points per encoded tricklet segment—see Figure 3. We illustrate our approach with the ICDM 2010 challenge dataset [28], which is also used in our experiments below. In a preprocessing step, we train a dictionary and obtain 131 distinct atoms. We use 30 atoms for encoding each tricklet segment. We set the size of the atoms to 1440 time points at 1 minute interval, which is equivalent to one day of data. The dictionary associated matrix has thus $131 \times 1440 = 188640$ cells. Figure 3 illustrates this process. Incrementally, our system takes as input one batch of tricklet segments (*Raw Tricklet Segments* in Figure 3). This table keeps only the timestamp and its corresponding value. We represent the first and last segments from the batch using colored rectangular areas. The dictionary that we use stores the atoms per column (*Dictionary* in Figure 3). For the sparse representation computed via a matching pursuit method (*Sparse Representations* on Figure 3), we store the tricklet segment timestamp, the ID of the atom used by the matching pursuit, the atom rank among the dictionary atoms, and the corresponding coefficient (i.e., the coefficient $w_i$ in Equation 1). As described in equation (1), the signal at a point $t$ is reconstructed as follows:

$$S = \sum_{i=1}^{k} w_i \alpha_i = \sum_{j \in SP} w_j \alpha_{i_j}$$

where $w_i$ is the coefficient value for atom $i_j$ at rank $j$ and $i_j$ represents the index in the dictionary of the $j$th atom with a non-zero coefficient. Only $sp$ non zero coefficients need to be stored in the database corresponding to the set: $SP = \{j_1, ..., j_{sp}\}$, where $j_k \in \{j_1, ..., j_k\}$ for $j_{k_1} \neq j_{k_2}; k_1 \neq k_2; k_1, k_2 \in \{1..k\}$. The upper bound $sp$ defines how many atoms are needed to reconstruct the signal without loss of precision.

There are two parameters that can be tuned during the dictionary learning phase and that are important for the quality of the stored results: the sparsity factor $sp$, and the number of dictionary atoms $k$. Many factors can impact what is right in terms of choice of these two parameters (accepted approximation degree in the signal reconstruction, time constraints, characteristics of datasets used in the training phase, domain of application, etc.). In this paper, our main focus is rather on the design of a system enabling efficient and effective analytics on massive amounts of tricklets, including the overall architecture of our system, its processing pipeline, its compression and storage layers as well as its dedicated query execution strategies.

To perform *analytics on uncompressed data*, all values are aggregated and grouped according to the desired dimension in time (e.g., hourly, daily, monthly, etc). The drawback of executing queries on uncompressed data is that the table contents must be scanned until finding the right time region (in the worst case, the complete table must be scanned). One improvement in terms of time can be done by using range queries; However, this solution would not perform well once the selectivity of the queries is high (see our experimental results in Section V). Misaligned and missing values aspects are easily captured by the possibility of usage of any elastic comparison methods (LCSS, DTW, to mention a few) in the matching pursuit process in the comparison step between the query and an atom.

When executing *analytics on compressed data*, there are two key steps to be considered: first, all relevant compressed entries are identified, all coefficients based on the desired accuracy are aggregated and grouped by atom id, and second, the result table is combined with the dictionary matrix. As each atom in the sparse representation table corresponds to a specific column in the dictionary matrix, this operation is typically carried out on the application server. Thanks to the hybrid nature of HYRISE, we can however directly store the dictionary matrix next to the relational table. This allows us to implement an indirect scalar multiplication as a plan operation. Compared to traditional database execution schemes, such an approach is closer in spirit to late-materialization strategies[2] known from column-oriented DBMSs. Figure 4 highlights the principles of the compressed query execution. In the first step, the relevant atoms are identified (action marked by 1 in Figure

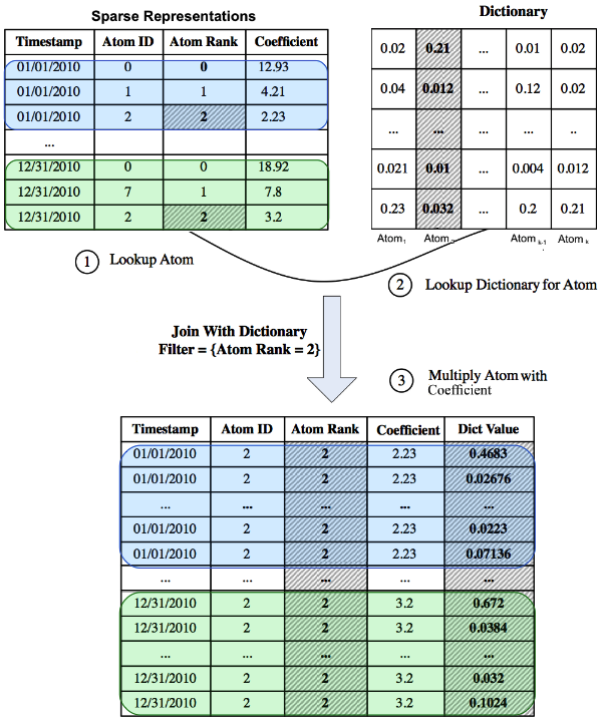


Fig. 4. Overview of query execution

4), and then retrieved from the dictionary (action marked by 2 in Figure 4). Then, depending on the aggregation level, the corresponding coefficients from the Sparse Representations table are multiplied by the corresponding atoms from the dictionary (action 3 in Figure 4).

The key idea for executing queries directly on the compressed representations is to take advantage of the linearity feature of the sparse representations. For instance, given a large number of tricklet segments $T_{B_1}, \ldots, T_{B_L}$ with their sparse representations $T_{B_l} = w_l * D$, querying for the sum of all these tricklet segments can be done as follows:

$$\sum_{l=1}^{L} T_{B_l}(t) = \sum_{l=1}^{L} \sum_{i=1}^{k} w_{j,l}\alpha_i(t) = \sum_{i=1}^{k} \alpha_i(t) \sum_{l=1}^{L} w_{i,l}.$$

Using this switch, the materialization for a given time point is deferred until the last phase of query execution, which is a major improvement over querying uncompressed data. However, not only is the decompression deferred until the last phase, but also the aggregation based on the atom ID is basically replacing a series of several additions with simple multiplications. Instead of accounting for all distinct values in the uncompressed data, e.g., $(a + a + a + b + a + a + b)$, we rewrite this based on the dictionary to $(5a+2b)$ which is more efficient to execute.

### D. TRISTAN's Merits

Modern database systems do not typically consider sparse dictionary compression techniques. However, we believe that sparse representations have a number of particularly attractive properties for database systems, and that our system exhibits a number of key features that make it very attractive for many tricklets and Smarter Cities applications. We summarize some of its merits below.

**Universality:** First, our approach is very general and can be applied to a very large number of scenarios and problems; second, as the dictionary atoms are not constrained in shape or number, it is possible to learn a dictionary to fit any set of signals. Fourier and wavelets transforms are particular cases of dictionary-based sparse representations; consequently, TRISTAN provides a time series representation that is also a generalization of the Fourier and wavelets transforms.

**Insightful Compression:** The time series representation provided by TRISTAN is especially suited for signals presenting very repetitive, noisy or missing values, that are non-uniformly sampled, or misaligned. Due to the repetitiveness of human behaviors, most of the city sensor data inherit such repetitiveness. The other features mentioned here are also very characteristic either to the human behavior or to our current measurement devices. Therefore, a dictionary offers excellent support for processing these behaviors by allowing them to be accurately represented using a handful of bits only. Moreover, as the atoms provenance is not constrained, the dictionary can be dynamically enhanced to include additional atoms one may consider interesting (e.g., anomalies detection, failure detection, fraud detection, etc.) without the need to re-encode the previous representations; this translates into an adaptive and extensible compression.

**Insightful Segmentation:** Tricklets are often analyzed with respect to a given time interval implied by human or natural patterns (e.g., hourly, daily, or weekly patterns.) It thus makes sense to split the time series into subsequences that are processed individually, and then aggregated. This feature is directly supported by the dictionary representation, since the dictionary atoms can be matched to such patterns.

**Linearity:** Our representation being linear, it is possible to answer many queries on the sparse representation directly (as described in Section III-C), without needing the costly operation of reconstructing the original signals. This leads to significant performance advantages for queries that can exploit this linearity (see Section V for some concrete examples.)

**Exponential Decay:** A key feature of our sparse representation is the exponential decay of the reconstruction error. The matching pursuit methods have the big advantage that the approximation error decreases monotonically and its decay is exponential. This feature is extremely helpful as it allows to dynamically adapt the sparsity level to match new accuracy degrees or compression constraints. Theoretical proof and studies on this phenomenon can be found in [18].

**Approximate Query Results:** Last but not least, the exponential decay property described above keeps the energy concentrated in the first atoms. This means that the first elements of a sparse representation contain most of the signal's

information and that, depending on the query being issued and of the application, the remaining atoms of the sparse representation can be dynamically dropped at query-time to yield a reconstruction that is slightly less accurate but much more compact. We give a few examples of such approximate query results in Section V.

All these features make TRISTAN an excellent system for signal denoising and recovery, aggregation, classification, outlier detection, forecasting, and related complex analytics queries. The dictionary representation presents however one drawback related to the difficulty of updating its *entire* contents efficiently and in an online manner to reflect totally new patterns, thus making it unsuitable to storing and processing highly variable data.

## IV. TRISTAN's Extensions

In this section, we briefly discuss how we have extended TRISTAN to address the problems of misaligned data and missing values. Both extensions are achieved through an *interpolation of the dictionary* instead of the time series themselves. Modifying the dictionary to offer support to continuous-time signal representation is made possible by using spline interpolation, e.g., by using cubic splines on each of the dictionary atoms. Using this interpolation, any sparse approximation algorithm can then be applied in its original form on an arbitrarily sampled time series using the interpolated version of the dictionary.

In many applications, sensors can dynamically adapt their sampling rate to minimize their I/O and optimize their battery life, and the database system would need the capacity to cope with arbitrarily sampled tricklets. Dictionary-based sparse representations offer an elegant solution to handle such problems and to allow inserting sensor-generated data directly into the database.

Reciprocally, given a time-discrete sparse representation of a tricklet segment, it is possible to use continuous-time representations of the atoms (i.e., interpolated version of the dictionary atoms) to retrieve an interpolated value from the tricklet segment (this also offers a way of approximating values in the tricklet segment). As cubic splines (and more generally speaking all kernel-based interpolations) are linear, it is easy to show that interpolating the dictionary atoms and then reconstructing the tricklet segment based on these interpolated atoms yields the exact same result as if the time series were reconstructed first and then interpolated. This means that computationally costly methods (e.g., the cubic spline interpolations have a complexity of $O(n^2)$) can be applied at once to an entire tricklet segment by just transforming the dictionary.

This feature presents a high application potential as it allows querying for the value of the tricklet at any timestamp with low costs. The entire database that stores the dictionary atoms can be interpolated in constant time—independently of the database size; in other words, TRISTAN allows interpolating a tricklet segments database with a $O(1)$ complexity.

## V. Experimental Evaluation

In this section we present our experimental evaluation of TRISTAN. Our test system is a 4-way Intel Xeon 7560 with 440 GB of main memory running Linux with a 3.5 kernel. As persistence layer, we used HYRISE [12], [13], which is a freely available open-source database system[1].

The main dataset we used for our experiments is based on the ICDM challenge [28] and consists of a stream of energy consumption samples. In addition, we also used a publicly available dataset from RTE France[2] consisting of electrical power consumption data from 1996 to 2012. The temporal resolution of the ICDM data set is one sample per minute, while the resolution of the data from RTE is 1 event per 30 minutes. We considered daily behaviors and thus split the data into daily segments (tricklet segments). Thus, each atom of the ICDM dictionary contains $1440$ points, while the atoms from the RTE dictionary contain $48$ data points each.

We structured our experimental evaluation presentation into three parts. In the first part, we evaluate the quality of the dictionary and of the resulting compressed representation. The second part presents an analysis of the properties of the compressed data using sparse dictionary encoding, while, in the third part, we evaluate the query execution performance with varying parameters.

### A. Dictionary Training

The time needed for learning the dictionary atoms given a set of tricklet segments depends on several parameters that are dictionary or application dependent (e.g., residual error threshold, number of atoms, etc.). A dictionary can be learnt while coding several billion data points in a matter of hours in a Map-Reduce environment [26]. As our focus is not on the dictionary learning part (that we consider belonging to an offline preprocessing step), but on the analytics that run on top of the sparse dictionary representations, we focus on evaluating the quality of the dictionary from a compression/querying point of view, the quality of the storage of the compressed representations, and the effectiveness and the efficiency of query execution.

Figure 5 gives the Root Mean Square Error (RMSE) of both the RTE and ICDM datasets compressed using our sparse dictionary technique. The figure also gives the error using a Fourier-based representation, which yields much higher error rates than TRISTAN's compression technique.

### B. Compression Ratio

In this subsection, we discuss the storage requirements and the compression ratio achieved by TRISTAN compared to storing a non-compressed version of the data. For this purpose, we compute the storage requirements for the dictionary, for the compressed, and for the non-compressed tricklet segments in a number of batches. For simplicity, we ignore the space needed for temporary computations. The

[1] http://github.com/hyrise/hyrise

[2] http://clients.rte-france.com/lang/fr/clients_consommateurs/vie/vie_stats_conso_inst.jsp

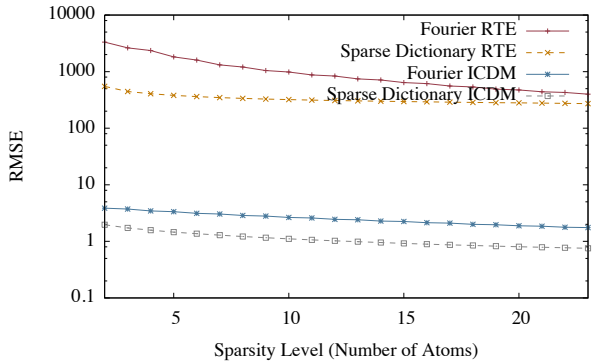


Fig. 5. RMSE comparison for different dictionary representations.

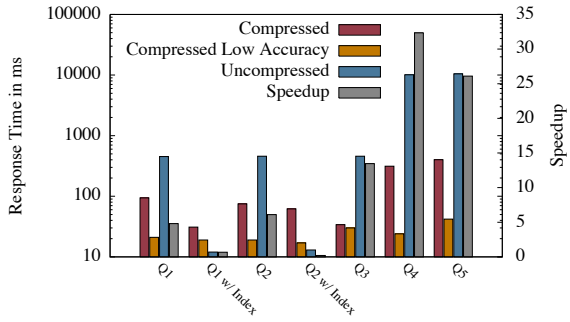


Fig. 6. Comparing compressed and uncompressed query execution times with low and high accuracy dictionaries on the ICDM data

compression ratio of our sparse compressed representation is mainly dependent on the number of atoms that are used to represent the original data. The dictionary needs to store the atoms (there are $k$ atoms), where each atom stores: *the value* ($\bar{n}$ values - a tricklet segments may have a variable number of elements that we represent by $\bar{n}$; each value is stored in one cell whose size depends on the data type, OS, programming language, etc), the *rank* (atom position among the atoms in the dictionary) and the *id* of the atom. Therefore, the dictionary size corresponds to: $s(D) = \bar{n} * k * sizeOf(value) + k * sizeOf(rank) + k * sizeOf(id)$. For storing $m$ compressed tricklet segments (i.e., $m$ sparse representations of the tricklet segments) over $b$ batches and using $k' \leq k$ atoms, we store for each batch and each tricklet segment: the $k'$ coefficients and their corresponding atom id and rank, plus one value for storing the id of the tricklet segment, summing up to a storage need of: $s(C) = m * b * (k' * sizeOf(coeff) + k' * sizeOf(rank) + k' * sizeOf(id) + sizeOf(id))$.
The compression ratio is estimated by making the ratio between the compressed and non-compressed version storage requirements: $\frac{m*b*(\bar{n}+sizeOf(id))}{s(D)+s(C)}$.

We use two different dictionaries for our experiments: one with 16 atoms (RTE) and one with 131 atoms (ICDM). The dictionaries are used to compress a single day of the original data sampled every minute. Hence, our system stores 1440 coefficient values for every dictionary atom. Due to the high oversampling rate of the ICDM data, the compression ratio is much higher compared to the RTE dataset. The compression ratio is 20 : 1 for the ICDM dataset, while it is 2 : 1 for the RTE data (which represents the worst-case scenario for our approach).

### C. Query Performance

We now focus on query performance and compare the performance of queries executed on the uncompressed and on the compressed data. To evaluate the performance of our system, we define the following workload:

- Q1 Select all values for a given day
- Q2 Aggregate the energy consumption for a specific day
- Q3 Aggregate the energy consumption for a period of 6 months and group by day
- Q4 Average yearly energy consumption over all recorded years.
- Q5 Variance of the energy consumption per month over all recorded years

We report the query execution times for the above workload running on our two data sets in Figure 6. On average, the speedup of TRISTAN over the uncompressed case is 12x for the high-accuracy reconstruction and 105x for the low-accuracy reconstruction. The results can be explained as follows: for low-selectivity queries (e.g., Q1 or Q2, which only retrieve data from a single day), little work has to be performed on the uncompressed data. In case no index is available for the timestamp values, the response time for the uncompressed case is dominated by the cost of scanning the input table to apply the predicate. If an index can be used to directly select the right set of records (see Figure 6 “Q1/Q2 w/ index”) the cost drops drastically. As for the compressed case, the cost of low-selectivity queries is dominated by the multiplication between the atoms and their corresponding coefficients. In such cases, all 1440 values have to be extracted from the dictionary for each atom appearing in the compressed signal, hence slowing down the reconstruction process.

The situation drastically changes for higher-selectivities (Q3-Q5). In such cases, the query execution benefits from pre-aggregating the selected atoms based on their atom ID and materializing their values from the dictionary as the last step of the execution plan. This processing strategy allows to achieve much higher speedups (e.g., 25x for the high-accuracy case, and 250x for the low-accuracy case on Q4)

The speedup is less impressive for the RTE data (see Figure 7), since the compression ratio is not as high in that case. However, for the large analytics queries we still observe a performance gain of almost 5x for the high-accuracy reconstruction, and of almost 25x for the low-quality reconstruction. Also, we observe that the difference between the low and the high-accuracy times is smaller in this case. This can be explained since we only use 6 atoms for the complete reconstruction of the RTE high-accuracy signal, compared to

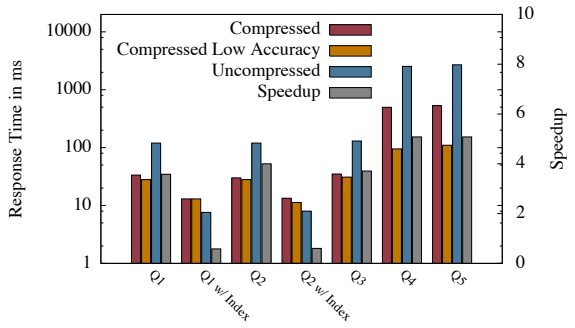


Fig. 7. Comparing compressed and uncompressed query execution times with low and high accuracy dictionaries on the RTE data set

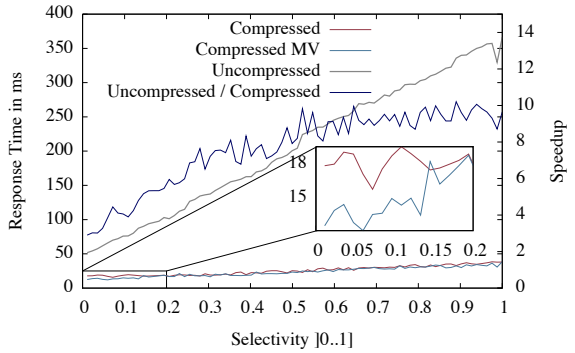


Fig. 8. Impact of selectivity on query execution with a dictionary of 130 atoms - ICDM

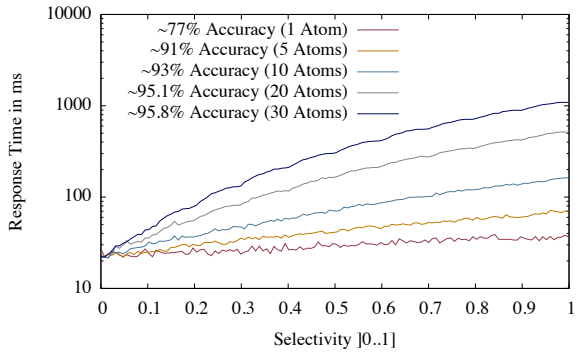


Fig. 9. Query execution with decreased accuracy and increasing selectivity – ICDM

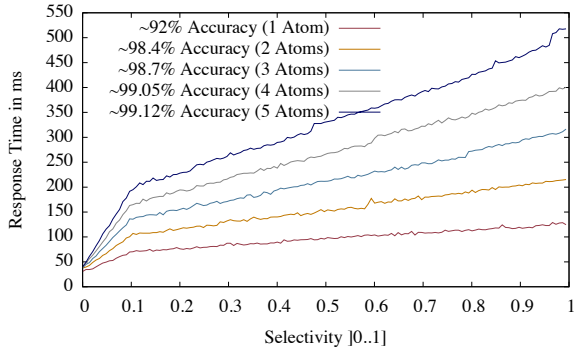


Fig. 10. Query execution with decreased accuracy and increasing selectivity – RTE

31 for the ICDM data

To further investigate the impact of selectivity on query performance, we modified query Q3 to select a varying timespan in order to gradually increase the selectivity of the query from 0 to 1. In addition to the standard dictionary, we created a materialized view of the aggregated dictionary values to avoid the frequent pre-aggregation rounds on all dictionary values. We think that such a materialization approach is realistic given that the dictionary is read-only during query execution. In the uncommon cases where the dictionary needs to be updated during query execution, the materialized view can be dropped and refreshed once the dictionary stabilizes. The results of this experiment are shown in Figure 8.

As described above in Section III-D, our dictionary compression techniques allow for approximate calculations of the results based on the exponential decay property. In order to analyze the impact of the number of atoms on query execution, we modified Q3 to vary the number of atoms. In addition, we modify the query in order to select a varying timespan in order to gradually increase the selectivity of the query from 0 to 1. As expected, the query runtime increases for a complete reconstruction of the signal as seen in Figure 9 and Figure 10. Since the exponential decay is known in advance, TRISTAN can return approximate query results with a specific error. Compared to uncompressed query execution, this allows to interactively explore large data sets, and to increase the accuracy of the results once the desired patterns are identified.

To evaluate the scalability of our system, we increase the size of the RTE dataset by multiplying the input data volume by a factor of 100. The original time series data is increased to about 300M records and the compressed time series data volume is increased to 44M records by this procedure.

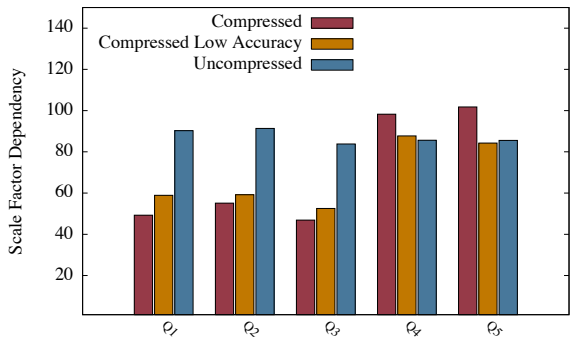


Fig. 11. Scalability experiment, comparing the query execution times on a 100x bigger RTE dataset.

We rerun the benchmark for this scaled dataset. We can observe on Figure 11 that the query execution times scale linearly with the scale factor. For the low-selectivity queries, the results are sub-linear since the amount of processed data does not increase linearly with the scale factor in such cases.

In summary, TRISTAN achieves high compression ratios on time series data and allows for very flexible query execution strategies, where one can trade accuracy for increased performance (e.g., for interactive exploration of Big Data.) Queries in our system are up to 250x faster on the sparse and compressed representation than on the uncompressed data, with an error rate known a priori thanks to the exponential decay property of our encoding technique.

## VI. Related Work

A number of approaches have already been proposed to efficiently process time series data, some tackling this problem from a pure data mining angle, others from a signal-processing angle. We give below a short review of related work focusing on time series representation and dictionary encoding.

### A. Times Series Encoding & Representation

Obtaining a new representation of a time series presents several advantages; a different representation space may be more compact, or may offer improved comparison or knowledge extraction capabilities. The most important techniques in that context are: Singular Value Decomposition, Fourier transforms and their relatives (FFTs, DCTs, Chebyshev, or Karhunded-Loeve transforms), polynomial approximations (Piecewise Linear Approximation, Adaptable Piecewise Constant Approximation), symbolic representation [7], [24], feature extraction (Kernel Piecewise Constant Approximation), wavelet transforms, probability distribution functions, and dictionary-based representations. We briefly discuss such representations below.

The first transformation often applied on the signal is due to the sampling rate, since the rate at which the values get registered by the sensors may not correspond to the sampling rate needed for the application. Techniques like downsampling, up-sampling interpolation, or decimation are often applied in that context. All those techniques do not preserve the peaks of the signal as they typically smooth it.

Wavelets transforms are among the most frequently used transformations for signal compression. Unlike Fourier Transforms, they provide both time and frequency information, yield good compression rates, and are good at representing signals exhibiting discontinuities or sharp peaks. They support both stationary and non-stationary signals and offer good support for trends, discontinuities in higher derivatives, and self-similarity [25] analyses. The Discrete Wavelet Transform (DWT) is a very popular wavelet transform, though it is not without its own limitations: The DWT is not shift invariant, has poor directionality, and lacks phase information [10].

Symbolic representation transforms the original signal into another signal whose elements are mapped onto symbols from a dictionary. One such technique is proposed in [7], [24] where, in a first step, the signal is cut into segments of equal sizes, followed by vertical segmentation of the area corresponding to each segment and finally mapping onto dictionary symbols.

A promising approach that combines several of the above techniques was presented in Reeves et al. [22]. Their approach decomposes the time series into low frequency parts, spikes (which are assumed to be sparse), and high frequency residuals. The low-frequency parts are captured using a low pass filter and downsampling, the spikes are stored explicitly, while the residual signals are compressed using random projections. This approach relies on the assumptions that spikes are relatively sparse and that most of the relevant aspects of the signal are in the low frequencies. In many real-world applications, however, this is not the case; Spikes can occur frequently and the high frequency parts of the signal carry important information that both Fourier decomposition and random projections cannot account for. We propose instead an unbiased approach that can represent the spikes, the low frequency and the high frequency components of a signal in a single formalism that still allows very efficient query processing on the compressed data.

### B. Dictionary Compression of Time Series

A method for executing similarity queries on compressed time series is proposed in [20]. In this work, a multi-resolution symbolic representation is computed for each time series. All time series are divided into segments of equal size, which are encoded in a dictionary. The new compressed representation of the time series is then a sequence containing only the frequencies of apparitions of the segments. The time series symbolic representation is thus unaware of any domain information. Also, the method does not propose optimized database storage solutions for the compressed time series and focuses on similarity queries only.

A Limpel-Ziv dictionary-based compression method for long-running time series is proposed in [16]. Time series classification based on dictionary-compressed time series is proposed in [5]; In this work, the dictionary contains a minimally redundant set of patterns to classify the time series, and is built by ranking patterns with respect to their utility for data classification. Both methods focus on similarity queries only do not offer analytics features.

Tricklets often contain a large amount of redundant information. Discarding the redundant information and keeping a representation that encodes with as few bits as possible the relevant elements presents many advantages (e.g., faster comparisons and less storage overhead). The signal-processing community has pioneered sparse dictionary representations focusing on compressing a signal using a linear combination of a *few* elements of a base dictionary only. There exists a vast body of literature [3], [23], [15], [17], [14] explaining how to determine a suitable dictionary for a given class of signals.

Optimally encoding a signal given a dictionary is a discrete optimization problem that is known to be $NP-hard$ and that was extensively studied [9], [21]. There exist two schools of

methods for solving this optimization problem: i) relaxation-based methods that transform the discrete optimization problem into a continuous one, and ii) matching pursuit-based methods [8] that use a greedy heuristic to quickly find a nearly optimal solution. TRISTAN uses the latter method.

## VII. Conclusions

In this paper, we proposed TRISTAN, a new system specialized in efficient storage and management of very large time series data. Our system exploits sparse dictionary representations for effective compression, compact storage, and efficient query execution over the compressed data. We believe that sparse representations have a number of particularly attractive properties for database systems handling time series. First, the approach is general and can be applied to a large number of scenarios and problems; the dictionary elements being not constrained in shape or number, it is possible to learn a dictionary to fit any set of signals. Second, the representation being linear, it is possible to answer a number of queries on the sparse representation directly without needing the costly operation of reconstructing the original signals. This leads to significant speed ups for queries that can take advantage of linearity as we showed in our experimental evaluation. Third, sparse representations have a reconstruction error that decreases exponentially with the number of elements used in the representation, meaning that the first elements of a sparse representation contain most of the signal's information and that, depending on the situation, the remaining atoms of the sparse representation can be ignored to yield a representation that is slightly less accurate but much more compact. We evaluated TRISTAN's efficiency and effectiveness experimentally on two real-world datasets and showed that TRISTAN can achieve up to 20:1 compression ratios and 250x speedups for query execution compared to traditional physical storage schemas. As future work, we plan to investigate how to extend TRISTAN to handle further data types and to provide a workload-driven, holistic compression technique to further reduce query execution times while maintaining low error rates.

## References


[1] D. J. Abadi, Y. Ahmad, M. Balazinska, M. Cherniack, J. hyon Hwang, W. Lindner, A. S. Maskey, E. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik. The design of the borealis stream processing engine. In *In CIDR*, pages 277–289, 2005.

[2] D. J. Abadi, D. S. Myers, D. J. DeWitt, and S. Madden. Materialization Strategies in a Column-Oriented DBMS. In *ICDE*, pages 466–475. IEEE, 2007.

[3] M. Aharon, M. Elad, and A. Bruckstein. K -svd: An algorithm for designing overcomplete dictionaries for sparse representation. *Signal Processing, IEEE Transactions on*, 54(11):4311 –4322, nov. 2006.

[4] L. Bar and G. Sapiro. Hierarchical dictionary learning for invariant classification. In *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, pages 3578–3581, March.

[5] Y. C. Bing Hu and E. Keogh. Time series classification under more realistic assumptions. *SIAM Conference on Data Mining*, 2013.

[6] T. Blumensath and M. Davies. A fast importance sampling algorithm for unsupervised learning of over-complete dictionaries. In *Acoustics, Speech, and Signal Processing, 2005. Proceedings. (ICASSP '05). IEEE International Conference on*, volume 5, pages v/213 – v/216 Vol. 5, march 2005.

[7] A. Camerra, T. Palpanas, J. Shieh, and E. Keogh. isax 2.0: Indexing and mining one billion time series. In *Proceedings of the 2010 IEEE International Conference on Data Mining*, ICDM '10, pages 58–67, Washington, DC, USA, 2010. IEEE Computer Society.

[8] M. E. Davies and T. Blumensath. Faster and greedier: algorithms for sparse reconstruction of large datasets. In *in Proceedings of the third International Symposium on Communications, Control and Signal Processing (ISCCSP*, 2008.

[9] G. Davis and M. Avellaneda. Adaptive greedy approximations.

[10] F. Fernandes, R. van Spaendonck, and C. Burrus. Multidimensional, mapping-based complex wavelet transforms. *Image Processing, IEEE Transactions on*, 14(1):110–124, Jan.

[11] L. Golab and T. Johnson. Consistency in a stream warehouse. In *CIDR*, pages 114–122. www.cidrdb.org, 2011.

[12] M. Grund, P. Cudré-Mauroux, J. Krüger, S. Madden, and H. Plattner. An overview of HYRISE - a Main Memory Hybrid Storage Engine. *IEEE Data Eng. Bull.*, 35(1):52–57, 2012.

[13] M. Grund, J. Krüger, H. Plattner, A. Zeier, P. Cudré-Mauroux, and S. Madden. HYRISE - A Main Memory Hybrid Storage Engine. *PVLDB*, 4(2):105–116, 2010.

[14] Z. Jiang, Z. Lin, and L. Davis. Learning a discriminative dictionary for sparse coding via label consistent k-svd. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1697 –1704, june 2011.

[15] Z. Jiang, Z. Lin, and L. S. Davis. Learning a Discriminative Dictionary for Sparse Coding via Label Consistent K-SVD. In *CVPR*, 2011.

[16] W. Lang, M. Morse, and J. Patel. Dictionary-based compression for long time-series similarity. *Knowledge and Data Engineering, IEEE Transactions on*, 22(11):1609 –1622, nov. 2010.

[17] J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online learning for matrix factorization and sparse coding. *J. Mach. Learn. Res.*, 11:19–60, Mar. 2010.

[18] S. Mallat and Z. Zhang. Matching pursuits with time-frequency dictionaries. *Signal Processing, IEEE Transactions on*, 41(12):3397–3415, Dec.

[19] A. Marascu, P. Pompey, E. Bouillet, O. Verscheure, M. Wurst, M. Grund, and P. Cudré-Mauroux. Mistral: An architecture for low-latency analytics on massive time series. In *Big Data and Smarter Cities*, pages 15–21, 2013.

[20] V. Megalooikonomou, Q. Wang, G. Li, and C. Faloutsos. A multiresolution symbolic representation of time series. In *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, pages 668 – 679, april 2005.

[21] B. Natarajan. Sparse approximate solutions to linear systems. *SIAM Journal on Computing*, 24(2):227–234, 1995.

[22] G. Reeves, J. Liu, S. Nath, and F. Zhao. Managing massive time series streams with multi-scale compressed trickles. *Proc. VLDB Endow.*, 2(1):97–108, Aug. 2009.

[23] R. Rubinstein, M. Zibulevsky, and M. Elad. Efficient Implementation of the K-SVD Algorithm using Batch Orthogonal Matching Pursuit. Technical report, CS Technion, Apr. 2008.

[24] J. Shieh and E. Keogh. isax: Indexing and mining terabyte sized time series, sigkdd. pp, 2008.

[25] M. Sifuzzaman, M. Islam, and M. Ali. Sparse and shift-invariant representations of music. *Journal of Physical Sciences*, 13:121–134, 2009.

[26] V. Sindhwani and A. Ghoting. Large-scale distributed non-negative sparse coding and sparse dictionary learning. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '12, pages 489–497, New York, NY, USA, 2012. ACM.

[27] K. Skretting and K. Engan. Recursive least squares dictionary learning algorithm. *IEEE Transactions on Signal Processing*, pages 2121–2130, 2010.

[28] M. Wojnarski, P. Góra, M. S. Szczuka, H. S. Nguyen, J. Swietlicka, and D. Zeinalipour-Yazti. IEEE ICDM 2010 Contest: TomTom Traffic Prediction for Intelligent GPS Navigation. In *ICDM Workshops*, pages 1372–1376. IEEE Computer Society, 2010.

[29] Q. Zhang and B. Li. Discriminative k-svd for dictionary learning in face recognition. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2691–2698, June.

[30] Y. Zhu and D. Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. In *In VLDB*, pages 358–369, 2002.