The VLDB Journal
The final publication is available at Springer via http://dx.doi.org/10.1007/s00778-014-0368-8.

A Time-Series Compression Technique and its Application to the Smart Grid

Frank Eichinger · **Pavel Efros** · **Stamatis Karnouskos** · **Klemens Böhm**

Received: 25 April 2013 / Accepted: 16 July 2014

Abstract Time-series data is increasingly collected in many domains. One example is the smart electricity infrastructure, which generates huge volumes of such data from sources such as smart electricity meters. Although today this data is used for visualization and billing in mostly 15-min resolution, its original temporal resolution frequently is more fine-grained, e.g., seconds. This is useful for various analytical applications such as short-term forecasting, disaggregation and visualization. However, transmitting and storing huge amounts of such fine-grained data is prohibitively expensive in terms of storage space in many cases. In this article, we present a compression technique based on piecewise regression and two methods which describe the performance of the compression. Although our technique is a general approach for time-series compression, smart grids serve as our running example and as our evaluation scenario. Depending on the data and the use-case scenario, the technique compresses data by ratios of up to factor 5,000 while maintaining its usefulness for analytics. The proposed technique has outperformed related work and has been applied to three real-world energy datasets in different scenarios. Finally, we show that the proposed compression technique can be implemented in a state-of-the-art database management system.

1 Introduction

Time-series data is one of the most important types of data, and it is increasingly collected in many different domains [34].

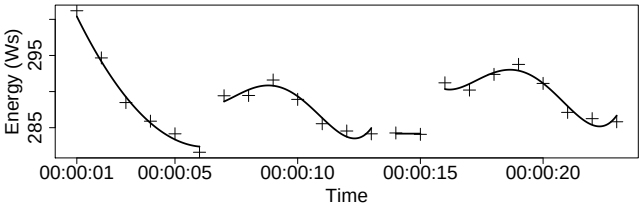


Figure 1 Example piecewise compression using regression functions.

In the domain of electrical energy, the advent of the *smart grid* leads to rapidly increasing volumes of time-series data [8]. This includes data from *smart electricity meters* and typically describes energy generation or consumption, but can also refer to other measurements such as frequency, voltage, electrical current etc. Such energy data serves as the foundation for many smart-grid applications and services [17] such as *forecasting* [4]. It can be used, for instance, for *demand response* [8] and energy trading [16]. However, while smart meters typically can generate data in high temporal resolution (e.g., measurements are recorded every second), frequently only measurements in a relatively low resolution (e.g., every 15 minutes) are being transmitted and used. At the same time, high-resolution data would be extremely beneficial for many analytical applications such as detailed visualization [30], energy disaggregation [21], monitoring of residential power quality [15], short-term forecasts of energy generation/load [7,33] or energy prices [1]. The explanation for not working with high-resolution data is the enormous amount of storage space required. For utility companies serving millions of customers, storing smart-meter measurements of high resolution would sum up to petabytes of data [8]. Even if storage prices are expected to decrease, the benefits of analytical applications will not always justify huge investments in storage. Similar concerns hold for the transmission of such data, which might cause congestion.

Work partly done while F. Eichinger and P. Efros were with SAP AG.

Frank Eichinger · Pavel Efros · Klemens Böhm
Karlsruhe Institute of Technology (KIT), Germany
E-mail: {eichinger, pavel.efros, klemens.boehm}@kit.edu

Stamatis Karnouskos
SAP AG, Karlsruhe, Germany
E-mail: stamatis.karnouskos@sap.com

To deal with such huge amounts of fine-grained time-series data, one possibility is to employ compression techniques. In contrast to *lossless* compression techniques [35] such as [13,45], *lossy* compression promises to achieve much higher compression ratios [36]. In the energy domain, data compression could be done directly in the smart meter, in a database management system or at a higher-level system such as an energy management system, a high-performance metering-data system etc. When doing lossy compression of data, one has to ensure that the original data can be retrieved or reconstructed with sufficient quality for the respective applications. This is, tasks like precise short-term load forecasting can only be done if very-close-to-original measurement values can be reconstructed from the compressed data. This requires guarantees regarding the maximum deviation of the values. Although time-series data and lossless compression have been investigated for quite some time, there is little research on lossy compression and retrieval of such data with guarantees. Studies such as [9,22,31] do not compress all types of data well or have disadvantages in terms of runtime. Moreover, none of these articles propose methods estimating the performance of their compression techniques.

This article presents a general lossy compression technique for time-series data from arbitrary domains. It builds on piecewise regression and learns polynomial regression functions of different degrees (see Figure 1 for an illustration). The compression technique guarantees that a user-defined maximum deviation from the original values is not exceeded for any point in the compressed time series. Besides energy data, the proposed technique is much broader and works with time-series data stemming from arbitrary domains, possibly being processed as *data streams*. In principle, data at all sampling rates can be used. The technique proposed can be executed on devices with limited computation capabilities such as smart electricity meters or in a database. The implementation that accompanies this article as well as the respective evaluation, are done in a database. In this article, we investigate the trade-off between accuracy, compression and performance. We do so by means of extensive experiments on real-world energy data from different sources. The technique achieves compression ratios of a factor of several thousands in certain scenarios and performs better than existing approaches in most cases. Naturally, compression ratios depend to a high degree on the data and the use-case scenario including the user-defined maximum deviation: In a load-forecasting scenario, results show that we can compress the data by a factors up to 17 without affecting forecasting results on most error metrics and up to factor 50 when a small decrease of accuracy is acceptable. In a price-estimation scenario on an energy-consumption dataset, we achieve compression ratios of more than factor 5.000. We additionally present two methods which estimate how well certain compression techniques perform. These methods can assist in deciding which type of compression is more appropriate for a given dataset. We also present an alternative approach for the compression of energy data, which makes use of the fact that there is a typical repeating pattern of energy consumption. We show and discuss how this approach can – in combination with our piecewise-regression technique – compress certain types of data even better than our piecewise-regression technique in isolation. Furthermore, to demonstrate one of the most important deployment scenarios of our compression technique, we implement our approach in the in-memory relational database management system SAP HANA and present some insights in this article.

Organization of the article: Section 2 discusses related work. Section 3 presents our compression technique. Section 4 describes our model of compression performance. Section 5 features an evaluation of our compression technique and of our model of compression performance. Section 6 contains further experimental results. Section 7 presents the deployment in a database management system. Section 8 concludes.

2 Related Work

Time-series data is of importance in many domains [34], and it has been investigated for years [4,19]. In this paper, we study *time-series compression*. As such, this topic has not received a lot of attention from the research community. However, established compression techniques can be applied to time-series data (Section 2.1). Furthermore, several data-management and data-mining techniques for time series, such as indexing, clustering and classification [19], provide means for compressed *time-series representation* (Section 2.2). Another topic, *time-series forecasting* (Section 2.3), has gained much attention from the research community [1,7,33]. As time-series forecasting is very important in smart electricity grids [8], we use it as one important scenario in our evaluation.

2.1 Compression Techniques for Time Series

There are two main categories of compression techniques: *lossless compression* and *lossy compression* [36]. With *lossless compression*, the full and exact reconstruction of the initial dataset is possible. Thus, lossless compression techniques are appropriate, for the compression of text documents or source code, to give examples. With *lossy compression*, some details of the data may be discarded during the compression process, so that the original dataset cannot be fully recovered from the compressed version. Such data losses can however be tolerable to a certain extent for

certain applications, such as picture, video and audio reproduction. JPEG, MPEG-4 and MP3 are popular examples of lossy compression techniques for these domains, respectively. We also hypothesize that many analytical smart-grid applications can tolerate certain losses. In the following, we discuss *lossless* techniques for the compression of time series. We then discuss *lossy* techniques in the context of *time-series representations* in Section 2.2.

The field of research on lossless compression includes established techniques, such as [13,45]. They are based on the reduction of the statistical redundancy of the data. However, none of these has been developed with the explicit goal of compressing time series. Recently, a study has investigated the applicability of lossless compression techniques on smart-meter data [35]. Next to specialized algorithms for devices with limited computing capabilities in sensor networks, the authors have studied a number of standard algorithms. They have conducted experiments on data measured each second; some of the datasets are very similar to the ones used in our evaluation. The authors have achieved the highest average compression ratio of factor 4 for datasets describing smart-meter readings from individual buildings using the bzip2 algorithm [36]. On metering data from individual devices, the authors have achieved the highest average compression ratio by a factor of 29 using the LZMA algorithm, an improved version of LZ77 [45]. This increase in compression ratio when compressing data of individual devices is natural, as individual devices typically consume the same amount of energy or even no energy for certain periods – such data containing few variations can be compressed more easily than data with higher variation. For standard smart-meter data, we expect lossy compression as investigated in this paper to achieve compression ratios which are orders of magnitude better. This certainly depends on the dataset and the quality requirements of the application. In certain situations where the original data needs to be fully reconstructible, lossless compression may however be better than no compression.

2.2 Time-Series Representations

In the wide field of time-series research, in particular regarding data management and data mining, several techniques have been proposed that can be used to generate an abstract representation of time series [5,10,18,20,24,38,39,43]. This includes Fourier transforms, wavelets, symbolic representations and piecewise regression. Some other related studies have investigated the representation of time series with the dedicated goal of reducing its communication or storage requirements [9,22]. All these techniques lead to time-series representations or abstractions which are usually smaller in size than the original time series. As they cannot be used to fully reconstruct the data, they are *lossy compression techniques*.

Discrete Fourier transforms have been used in [10] to extract features from time series. This helps to build an efficient subsequence-matching algorithm for time series. [5] is an approach for a similar time-series-matching problem, but it builds on *discrete wavelet transforms*, particularly *Haar wavelet transforms*. [38] proposes a wavelet-based tree structure to support certain queries on time-series data.

Fourier transforms and wavelets have been applied in these contexts and provide a means of representing and compressing time-series data. However, these techniques do not provide absolute guarantees for a compressed point. This would be necessary for time-series-based applications in many domains including smart electricity grids.

Another approach for time-series compression is symbolic data representation, which is based on discretization [24,39]. This has recently been applied to smart-meter data [43]. Symbolic representations can lead to very high compression ratios, but the compressed data can only "support statistics and machine learning algorithms for some selected purposes" [43] – many of the applications we have mentioned earlier cannot be executed on such highly compressed data.

Piecewise Regression

Piecewise-regression techniques divide time series into fixed-length or variable-length intervals and describe them using regression functions. As regression functions, all types of functions can be used in principle. However, polynomial functions, particularly constant and linear functions, can be estimated efficiently and are used frequently. [44] employs constant functions that approximate fixed-length intervals. [18,22] work with constant functions, too, but consider variable-length intervals. This gives the algorithm more flexibility to find a good compression model with the regression functions. All techniques mentioned can provide quality guarantees under the *uniform norm* (also called L_∞ *norm*). A quality guarantee under this norm means that any value of a decompressed time series deviates by less than a given absolute value from the corresponding one of the original time series.

[9] introduces two time-series compression techniques which produce connected as well as disconnected piecewise straight-line segments of variable length with a quality guarantee under the uniform norm. [20] is a similar approach with extensions that construct weight vectors which are useful for certain data-mining tasks such as classification and clustering.

Although their performance is good, our experiments with constant and linear piecewise polynomial regression (we have used [9,22]) show that these techniques cannot

compress highly variable data such as energy consumption data well (see Section 5.3.4).

The authors in [31] have presented the idea of employing not one, but several regression models for online time-series compression with a given quality guarantee. In its first step, their algorithm employs a set of regression models on a segment containing two points. Second, the algorithm stores the models that achieve an approximation of the segment under the given quality constraints in a temporary list. Third, the algorithm adds the subsequent incoming point to the segment and re-employs the regression models present in the temporary list. Fourth, the algorithm removes the models that did not succeed in approximating the newly formed segment within the given quality guarantees from the temporary list. The last three steps (the second to the fourth step) of the algorithm form a loop that repeats itself as long as the temporary list contains at least one model. Once the list is empty, the algorithm chooses the model which obtained the maximum value for the following ratio:

$$\frac{\text{length of longest segment approximated successfully}}{\text{number of parameters of the model}}.$$

The algorithm then saves the model and segment information (beginning and end points) instead of the actual data points. The algorithm restarts with its first step beginning with the segment of two points situated right after the segment just compressed.

By trying several models and choosing the best one, the weaknesses of each individual model are avoided, and a better compression ratio is achieved. However, [31] learns multiple regression models at each step of the approximation algorithm, which is disadvantageous in terms of runtime. Thus, although it might be beneficial to employ a larger number of different models in order to achieve higher compression ratios, the runtime of the algorithm grows linearly with the number of models employed.

Moreover, [31] does not propose any method to estimate compression performance. Such a method can help the database optimizer decide which technique it should use, given an application scenario and datasets. As an example, knowing that a simple compression technique, i.e., one based on constant functions, achieves the best compression on a given dataset, the optimizer can avoid employing other more resource and time consuming techniques.

In this article, we build on the idea of employing multiple models, but present a more efficient approach. We avoid employing multiple regression models at each step of the algorithm. We do so by relying on an incremental employment of polynomial regression models of different degrees. Moreover, by using polynomials of higher degrees, our approach handles highly variable data well. Internally, our algorithm employs three techniques [6,22,37] to approximate segments of time series using polynomial functions of different degrees. All of them provide guarantees under the uniform norm. Furthermore, we present two methods for estimating the compression performance. Our methods predict the storage space requirements of our compression technique.

Model Selection for Time-Series Prediction

Another approach for time-series compression in a different setting is presented in [23]. The authors consider savings in communication costs (and consequently energy consumption) in wireless sensor networks. The main idea is to estimate a model in a sensor node and communicate it to a sink. The sink then uses this model to 'reconstruct' subsequent sensor readings until the sensor node communicates a new model. In contrast to [31] and to our approach, [23] employs forecast models which predict future values of a time series instead of regression models. [31] and our approach have the advantage of instantiating their models based on the entire current segment of data. This typically results in better compression ratios.

Like [31] and in contrast to our approach, [23] combines multiple models in parallel. Concretely, the authors use autoregressive forecast models to compress the data by means of prediction. One disadvantage of using autoregressive models is that to reconstruct a given point of the time-series, all the values up to that point in time need to be reconstructed as well.

To reduce the number of models to maintain, the authors employ a racing strategy based on the statistical *Hoeffding bound* to pre-select the most promising models. As a result, after a certain period of time, the algorithm in [23] maintains only the model with the best prediction. However, in many settings the data may change its statistical properties with time. Examples from the energy domain would be the presence of an anomaly in the network or the deployment of a new device with a highly variable energy consumption. In this case, the model chosen using the racing mechanism would probably perform worse than other potential models which have been eliminated. Our approach in turn only maintains one model at a time and only switches to more complicated models if an easier model cannot compress the data well. Thus, our approach is not sensible to such situations, as it adapts to the data and chooses the model which best fits the current segment of data.

2.3 Time-Series Forecasting

Forecasting time-series data is a standard task for time-series data [4], and forecasting energy-related data is particularly important in the context of the *smart grid* [8]. In recent research, an impressive number of forecasting techniques has been developed particularly for energy demand [1,7, 33]. The author in [40] shows that the so-called *exponential*

smoothing technique behaves particularly well in the case of short-term energy demand forecasting. The main idea behind the *exponential smoothing techniques* is the representation of any point of the time series as a linear combination of the past points with exponentially decaying weights [25]. The weights are determined by *smoothing parameters* that need to be estimated. In the case of *triple exponential smoothing* (also called *Holt-Winters technique*), the time series is decomposed into three components: level, trend and season, each of which is modeled by a separate equation. We study the effects of our compression technique on forecasting based on this algorithm. In concrete terms, we will investigate the effect of using compressed data as input to triple exponential smoothing compared to using the original data.

3 A Piecewise Compression Technique

We now describe our compression algorithm (Section 3.1), discuss the selection of regression functions (Section 3.2), specify how to store compressed data (Section 3.3) and say how we compute compression ratios (Section 3.4).

3.1 An Approach for Time-Series Compression

Our piecewise regression technique employs a greedy strategy to compress intervals of a time series. This is necessary as the size of the state space for an optimal solution is extremely large. Internally, our technique uses three online regression algorithms providing guarantees under the uniform norm. Each one is specialized in one of the following classes of polynomial functions: constant functions (polynomials of degree zero), straight-line functions (polynomials of first degree) and polynomials of degree higher or equal than two. The PMR-Midrange algorithm [22] outputs the best approximation of a set of points using a constant function in $\mathcal{O}(1)$ time, by using the maximum and minimum values of the set of points at each of its steps. The algorithm introduced in [6] outputs the optimal approximation of a given set of points in maximum $\mathcal{O}(n)$ time using a straight-line function, with n being the number of points in the set. Finally, the randomized algorithm introduced in [37] calculates near-optimal approximations in $\mathcal{O}(n)$ time using a polynomial function of any degree.

The main algorithm (Algorithm 1) of our compression technique employs these three algorithms incrementally, and the compression result depends on a user-defined maximum tolerable deviation. This deviation is realized as a threshold on the uniform norm between the original and the approximated time series. This norm is defined as the maximum absolute distance between any pair of points of the real (x_i) and the approximated (x'_i) time series S: $L_\infty = \max_{i=1,\ldots,|S|} |x_i - x'_i|$.

Algorithm 1 Piecewise compression algorithm.

```
1:  Let p be the max. degree of polynomials to be used
2:  Let S be the time series for compression
3:  while |S| > 0 do
4:      current_seg = new_basic_length_segment(S)
5:      for k in 0 : p do
6:          while (approx_succeeded(k, current_seg)) do
7:              add_next_point(current_seg)
8:          end while
9:          save_polynomial(k, current_seg, aprox_params)
10:     end for
11:     choose_best_model_and_save()
12:     remove_segment(S, current_seg)
13: end while
```

In Algorithm 1, p corresponds to the maximum degree of the polynomials to be used within the algorithm. It is a user-defined parameter, and its appropriate value is to be defined based on preliminary experiments. We start with a segment of two points (Line 4) and loop over polynomial regression functions by their degree k, going from $k = 0$ to $k = p$ (Line 5). Depending on the value of k, we employ the corresponding specialized regression algorithm out of the three algorithms listed above. At each step, as long as the approximation of the current segment using the polynomial function of degree k attains the precision guarantee (Line 6), we add the next point of the time series to the current segment (Line 7). Once the precision is not attained any longer, we temporarily save the current polynomial parameters and the length of the segment the corresponding approximation had attained the precision guarantee for (Line 9). We then pass to the polynomial of the next degree and repeat the procedure (Lines 5 to 7). The loop terminates when we reach the polynomial of highest degree and when it cannot approximate the current segment with the requested precision any more. We then choose the polynomial that achieves the highest compression ratio (Line 11; see Section 3.4 for the calculation of the compression ratio). We compress the corresponding segment by saving its start and end positions, as well as the coefficients of the polynomial. The piecewise compression process just described then restarts beginning with the next segment.

Algorithm Analysis

In the following we analyze the runtime and memory usage of Algorithm 1. As stated above, our technique uses three online regression algorithms which provide guarantees under the L_∞ *norm*. We will first present an analysis of the runtime and memory usage of each of these algorithms. We will subsequently describe an equivalent analysis of Algorithm 1.

The PMR-Midrange algorithm [22] runs in $\mathcal{O}(1)$ per approximation step, i.e., for every incoming point. This is because it only needs to compare the value of the current

point with the minimum and maximum of the previous sequence of points. Moreover, it also needs $\mathcal{O}(1)$ memory, as it keeps the following three values for any sequence of points in memory: maximum, minimum and approximating value.

The algorithm for straight lines from [6] outputs the optimal approximation of a given sequence of points in $\mathcal{O}(n)$ time, with n being the number of points in the sequence. To do so, it uses several geometrical properties of the convex hull of the sequence of points. The algorithm keeps the convex hull of the sequence of points in memory. This takes at most $\mathcal{O}(n)$ space, but usually much less because only few points are part of the convex hull of the sequence.

Lastly, the randomized algorithm from [37] outputs near-optimal approximations using a polynomial function in $\mathcal{O}(d \cdot n)$ expected time, where d is the degree of the polynomial function and n is the number of points in the sequence. The main idea behind this algorithm is as follows: If n is bigger than d, most of the constraints of the linear programming problem associated with the approximation are irrelevant and can be discarded. The algorithm therefore chooses constraints at random and discards them until their number is small enough for the problem to be trivial and thus easily solvable. From the memory usage point of view, this algorithm occupies $\mathcal{O}(d \cdot n)$ space in memory during its execution.

Thus, on the one hand, Algorithm 1 runs in the least possible time ($\mathcal{O}(1)$) when it only uses the PMR-Midrange algorithm. On the other hand, if the approximation using constants or straight lines does not perform well enough, Algorithm 1 will often use the randomized algorithm and thus run in $\mathcal{O}(d \cdot n)$ expected time. However, our evaluation in Section 5 has shown that the worst-case scenario occurs rarely. From the memory usage perspective, Algorithm 1 needs as much as $\mathcal{O}(d \cdot n)$ space. When p is set to 0 or 1, Algorithm 1 consumes $\mathcal{O}(n)$ space necessary to keep the sequence of points in memory.

An advantage of our approach is that we do not deploy all compression schemes per approximation step. Thus, compared to the algorithm in [31], instead of deploying all compression schemes available, our algorithm starts by deploying only one scheme in each step and keeps deploying the same scheme as long as its approximation of the current segment succeeds under the given maximum-deviation constraint. Hence, we expect our compression technique to achieve better runtimes than related work.

Another advantage of the implementation of our compression algorithm is the distinction between the three classes of polynomials and the deployment of the respective optimal algorithms. At each step of the algorithm, the most appropriate technique is used. In particular, our technique starts by using the quickest algorithm (PMR-Midrange [22] running in $\mathcal{O}(1)$ time) and uses it as long as possible, changing to a slower algorithm only when the maximum-deviation constraint is violated. We hypothesize that this also results in high compression ratios and better runtime than related work.

3.2 Selection of Regression Functions

Our technique described in the previous section uses polynomial functions. We have chosen this type of function because of its simplicity and because their parameters can be estimated very efficiently. However, our technique can also employ any other type of functions, be it entirely, be it in addition. In preliminary experiments, we have tested the sine function due to its use in Fourier series decomposition within our algorithm. Our intuition was that the sine function might be able to fit longer segments of variable energy consumption, which would result in higher compression ratios. To estimate the sine functions with non-linear combinations of model parameters, we have used the algorithm described in [41].

The result of these preliminary experiments is that it does not provide any significant positive effect on the compression ratio (less than 1% improvement compared to the values given in Section 5.3). In fact, the compression ratio has sometimes been worse when using the sine function. At the same time, the execution time grows exponentially when the corresponding algorithm is used to approximate a long segment of points. We therefore choose to not consider this type of function any further. If further research reveals that functions other than polynomial ones might be suited to compress certain datasets, they can however be incorporated in our compression technique.

3.3 Storing Compressed Time-Series Data

To store the results of our compression technique and to have a basis for the calculation of compression ratios (see Section 3.4), we have to specify a data layout. In order to facilitate the access to and retrieval of the data, we have chosen to store the compressed data in a relational database system. We choose a database schema similar to the one used in [31] in order to use the same definition of compression ratio for comparison purposes. This database schema consists of one table to store the compressed intervals (Table 1). In order to communicate compressed data, a serialized version of this table can be used. The regression functions themselves can be hard-coded or can alternatively be stored in an additional table. We store the compressed intervals (*segments*) by saving the following values in *COMPR_SEG* (Table 1): the id of the time series or device that has captured the time series (*series*), the id of the regression function used to approximate the segment (*func*), the timestamps corresponding to

Table 1 *COMPR_SEG* – Example table for storing compressed segments.

series	*func*	*t_start*	*t_end*	*coefficients*
1	2	1339535405	1339538506	105.0, 0.4
2	4	1349639407	1349689309	102.3, 0.1, 2.7, 4.6

the start and the end of the segment (*t_start* and *t_end*) and the coefficients of the function (*coefficients*).

From this (or a similar) table structure, the original values can be reconstructed with a single declarative SQL query. This query accesses the relevant rows and directly calculates the values of the associated regression functions.

3.4 Calculation of Compression Ratios

The compression ratio is needed within our algorithm (Line 11 in Algorithm 1), and we use it as a criterion in the evaluation (Section 5.3). It is equal to the ratio between the size of the initial data and the size of the compressed data.

The compression ratio is obtained by firstly calculating the size of the initial uncompressed data. An uncompressed time series can be saved by storing its id, the time each value was captured at and the corresponding measurement value. Consequently, the size of the initial data is equal to the sum of the sizes of each reading in the respective table.

The size of the compressed data is calculated similarly by summing up the sizes of the compressed segments. The size of a compressed segment is equal to the sum of the sizes of each data type in Table 1 for the given segment. Finally, the compression ratio is calculated by dividing the initial size of the data by the size of the compressed data:

$$\textit{compression ratio} = \frac{\text{size of initial data}}{\text{size of compressed data}}.$$

The calculation of the compression ratio relies on the size of the initial and the compressed data. In our algorithm and in the experiments presented in Section 5.3, we rely on the storage scheme presented in Section 3.3. However, we assume certain storage requirements for the different data types. These may be adapted to the actual storage requirements in a realistic deployment of our technique, e.g., in a database management system. For the sake of simplicity, we assume a size of 64 bits for every data type in Table 1. As our compression algorithm uses the compression ratio as an internal optimization criterion (Line 11 in Algorithm 1), best compression ratios may be achieved by refining the formula given in this subsection with the actual values for storage requirements. We investigate this in more detail in Section 7.

3.5 Parameter Settings

Our technique has two parameters: the maximum deviation allowed and the maximum polynomial degree. The maximum deviation allowed is the error bound on the L_∞ *norm* the compression has to guarantee. This parameter depends on the application using the data (e.g., which deviation from the original data can the application tolerate and still run successfully). Section 5.2 says how to find suitable values for this parameter in smart-grid scenarios.

Regarding the maximum polynomial degree, our evaluation shows that a value of 3 is sufficient for any of our datasets and scenarios. This value can be overwritten, and this may boost performance in some settings. For instance, Section 5.3.1 shows that a value of 1 is sufficient for one of our scenarios. To assist with the choice of this parameter, a system administrator or a self-tuning component can use one of our methods to reliably estimate the compression ratio. The main benefit of lower values is that the compression takes less time and resources.

4 Estimation of Compression Ratio

In the following we propose two methods to determine the storage-space requirements of our compression technique. The first method, which we call *model-based estimation* in the following, uses a statistical model of the average length of the segments resulting from piecewise regression. This method can be instantiated with a broad range of piecewise regression techniques. In what follows, we illustrate this with two popular techniques as examples: the first one uses constant functions – the PMR-Midrange algorithm introduced in [22], the second one uses disconnected straight-line functions – the slide filter introduced in [9]. Note that constant functions are a special case of the latter, namely straight-line functions whose slope is zero.

The second method, which we refer to as *generation-based estimation* in the following, is generic in that it can be instantiated with any piecewise-regression technique. It efficiently generates a large number of time series. It then uses the average length of these time series as an estimate of the average length of segments resulting from the piecewise regression.

In the following we first describe each method in detail. We then present and evaluate their runtime performance. We present their experimental comparison in Section 5.5.

4.1 Model-Based Estimation

In this subsection, we first present the intuition behind the model-based method. We then explicitly describe how it works in the case of constant and straight-line functions.

4.1.1 Intuition

We propose a statistical model of the average length of the segments resulting from piecewise compression. To do so, we rely on the following observation: Many time-series, e.g., describing energy data, have high positive autocorrelation values. Modeling such a time series as a random variable and assuming that its samples are independent and identically distributed (i.i.d.) yields erroneous results. However, differencing a time series, i.e., calculating the series containing the differences between two consecutive time points, is a transformation, developed in [3], which tends to produce less autocorrelated time series. This also happens when applying this transformation to the time series in our datasets (Section 5.5). Table 2 shows the autocorrelation values for the original and first-difference time-series of two energy consumption datasets (see Section 5.1). We use the following formula [4] to calculate the autocorrelation of a time-series X_t with mean μ at lag h:

$$\rho_X(h) = \mathbb{E}[(X_{t+h} - \mu)(X_t - \mu)] \tag{1}$$

The table contains values at the first three lags. Further experiments of ours have yielded similar results for larger lags. The original time series have high autocorrelation values (> 0.95) for all lags shown in Table 2. At the same time, the autocorrelation values for the first-differences time-series are much smaller and close to zero. We conclude that there is much less correlation between consecutive values of the first-differences time-series.

Table 2 Autocorrelation values for original and first-difference time-series of energy consumption data.

		lag		
		1	2	3
REDD house 1	original	0.996	0.991	0.986
	differences	0.092	-0.040	-0.006
REDD house 2	original	0.986	0.971	0.957
	differences	-0.007	-0.005	0.007
Smart* home B	original	0.998	0.995	0.993
	differences	0.094	-0.018	-0.013
Smart* home C	original	0.980	0.968	0.959
	differences	-0.191	-0.077	0.030

To obtain an estimate of the average length of those segments, we model the distribution of the differences between consecutive values of the time series as a random variable D. We assume that data generation is equivalent to generating i.i.d. samples D_1, D_2, ..., D_n of D. Using this assumption, we model the length of the current segment with the given compression function (constant or straight-line function) as a random variable and calculate its expected value. This expected value is an estimation of the average length of a segment.

4.1.2 Constant Functions

The constant function with the minimum distance under the L_∞ *norm* to a given segment of a time series is defined by the value $\frac{max-min}{2}$, where *max* and *min* are the maximum and minimum values of the segment [22]. Recall that there is a predefined error bound ε on this distance, which the piecewise compression technique has to guarantee. The constant function can compress the segment of time series within the given error bound if the following condition is satisfied: $\frac{max-min}{2} \leq \varepsilon$ (Condition 1). As mentioned above, we model the first-difference time-series as a random variable D. We assume that the points of the time series are generated as samples of D. Summing up the first n samples $D_1, D_2, \ldots D_n$ results in the difference between the $(n+1)^{th}$ value and the first value of an original (non-difference) time series X_t:

$$\sum_{i=1}^{n} D_i = (X_1 - X_0) + (X_2 - X_1) + \cdots + (X_n - X_{n-1}) = X_n - X_0 \tag{2}$$

We can thus remap Condition 1. For a constant function to approximate a time series within a given error bound, the range of the partial sums of the current samples of D has to be smaller than or equal to $2 \cdot \varepsilon$:

$$max(S(D,n)) - min(S(D,n)) \leq 2 \cdot \varepsilon \tag{3}$$

where $S(D,n) = \{D_1, \ldots, \sum_{i=1}^{n} D_i\}$. Next, a random variable Z models the number of consecutive points generated by sampling D which a constant function can approximate, given an error bound ε. The probability of Z having a certain value is as follows:

$$\Pr(Z = n) = \Pr(max(S(D,n)) - min(S(D,n)) \leq 2 \cdot \varepsilon \wedge max(S(D,n+1)) - min(S(D,n+1)) > 2 \cdot \varepsilon) \tag{4}$$

The probability distribution of Z may be obtained in different ways. First, one can fit a well-known probability distribution, such as the Gaussian one, to the distribution of D. Second, one can estimate it directly by subsampling the data and calculating each probability using the samples. We use the latter option. The next step is calculating the expected value of Z:

$$\mathbb{E}[Z] = \sum_{i \geq 1} \Pr(Z > i) = \sum_{i \geq 1} \Pr(max(S(D,i)) - min(S(D,i)) \leq 2 \cdot \varepsilon) \tag{5}$$

To approximate $\mathbb{E}[Z]$ we use the following lemma, which is based on results from [12]:

Lemma 1 *Let $[Y_1, Y_2, \ldots, Y_n]$ be a sequence of mutually independent random variables with a common distribution Y. Let*

$S(Y,n) = \{Y_1, \ldots, \sum_{i=1}^{n} Y_i\}$

and let

$R_n = max(S(Y,n)) - min(S(Y,n))$.

Then the following holds:

$\mathbb{E}[R_n] = 2\sqrt{2n/\pi}$

Thus, the addends in Equation 5 decrease to 0 as i increases. Namely, as i increases, the range of $S(D,i)$ increases as well, and thus the probability that this range stays within $[0, 2 \cdot \varepsilon]$ decreases.

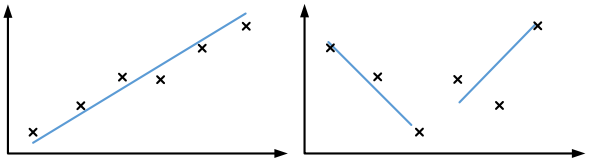
Algorithm 2, used to approximate $\mathbb{E}[Z]$, is explained next. We calculate the terms of the sum one by one (Line 8) until the current term is smaller than a given threshold δ (Line 6). Intuitively, δ should be set to a value much smaller than the value we expect for $\mathbb{E}[Z_{low}]$ and close to 0. Moreover, the lower δ, the better the approximation. As we expect the average length of the segment to be in at least the order of magnitude of 1, we set this threshold to 0.001, i.e., three orders of magnitude smaller.

Algorithm 2 Estimation for constant-value functions

```
1:  Let ε be the predefined error bound
2:  Let δ = 0.001
3:  Let i = 1
4:  Let add = Pr(Z > i)
5:  Let estimation = add
6:  while add > δ do
7:      add = Pr(Z > i)
8:      estimation = estimation + add
9:      i = i + 1
10: end while
```

4.1.3 Straight-Line Functions

Estimating the average length of the segments resulting from the piecewise compression using arbitrary straight-line functions is more complex. In the case of constant functions, the maximum and minimum of $S(D,n)$ determine whether the constant function can compress the current sequence of points, and the order of the samples of D does not play a role. In contrast to this, the order of the samples of D determines whether a straight line function can compress the sequence of values. Figure 2 illustrates this. There are two time series, with the same values, but these are ordered differently. The first time series is compressible using one straight-line function in its entirety, while the second time series needs two straight-line functions to be compressed within the same error bound.



(a) Time series 1 (b) Time series 2

Figure 2 Two time series with equal values but different orderings.

This makes a direct estimation as in the case of constant functions difficult, due to the exponentially high number of possible orderings of the samples of D. We therefore do not estimate the average length of the segments directly. Instead we recur to estimating lower and upper bounds of this length. More specifically, we underestimate the lower bound and overestimate the upper bound. We use the following notation:

- X_t, with $t \geq 0$, is the original (non-difference) time-series consisting of a sequence of samples from D.
- (t, X_t) are points in the two-dimensional space generated by the time axis and the other dimension being the range of the values.

Without loss of generality, we set X_0 to 0, and we fix the distance in time between two consecutive points of X_t to 1. Thus, we obtain:

$$\sum_{1}^{i} D_i = X_n - X_0 = X_n \tag{6}$$

Lower Bound: We now describe a model that lower bounds the number of points a straight line can approximate within a given error bound. The main idea behind our model is that we fix the first point the approximating straight-line passes through to $(0, X_0)$. Doing away with this degree of freedom narrows down the set of possible approximating lines. In other words, this lets us establish a lower bound.

Our model uses two sets of random variables: $LB_{up,i}$ and $LB_{low,i}$, with $i \geq 1$. At step i, based on the current sequence of samples $[X_0, \ldots, X_i]$, $LB_{up,i}$ is the smallest slope of all straight lines passing through the point $(0, X_0)$ and one of the points $(1, X_1 + \varepsilon), \ldots, (i, X_i + \varepsilon)$. The variable $LB_{low,i}$ is the largest slope of all lines passing through the point $(0, X_0)$ and one of the points $(1, X_1 - \varepsilon), \ldots, (i, X_i - \varepsilon)$. Intuitively, an approximating straight line must not have a slope bigger than $LB_{up,i}$ or smaller than $LB_{low,i}$. Otherwise the straight line would be too far away from one of the points, given the error bound ε and the above-mentioned limitation of our lower-bound model. Figure 3 graphs the first three samples X_0, X_1 and X_2, as well as the lines with slopes $LB_{up,1}$, $LB_{up,2}$, $LB_{low,1}$ and $LB_{low,2}$. We calculate these variables for $i \geq 2$

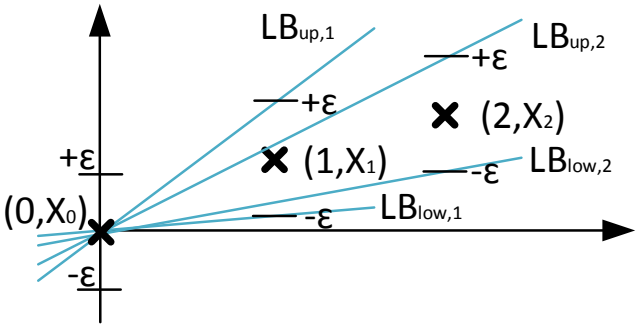


Figure 3 Estimation using straight-line functions – lower bound.

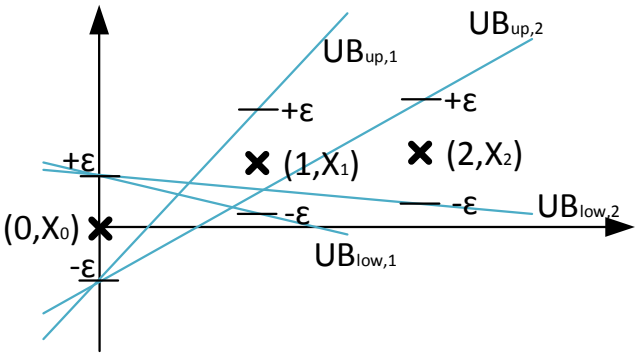


Figure 4 Estimation using straight-line functions – upper bound.

using the following recursive formulas:

$$LB_{up,i} = min(LB_{up,i-1}, \frac{X_i + \varepsilon}{i}) \quad (7)$$

$$LB_{low,i} = max(LB_{low,i-1}, \frac{X_i - \varepsilon}{i}) \quad (8)$$

with

$$LB_{up,1} = X_1 + \varepsilon$$
$$LB_{low,1} = X_1 - \varepsilon$$

The necessary and sufficient condition for our lower-bound model to be able to approximate the sequence $[(0,X_0), \ldots, (i,X_i)]$ within the predefined error bound ε is:

$$LB_{up,i} \geq LB_{low,i} \quad (9)$$

If this condition does not hold, the approximation fails. The following lemma, whose proof is in the appendix, shows that our model is a lower bound for the average length of the segments:

Lemma 2 *Let a sequence of data points* $[(t_0,X_0), (t_1,X_1), \ldots, (t_m,X_m)]$ *and an error bound* ε *be given. If* $LB_{up,m} \geq LB_{low,m}$*, then there is a straight line approximating the sequence of data points within the given error bound.*

Next, we determine the expected value of the number of consecutive points our lower-bound model can approximate – the random variable Z_{low}. The probability of Z_{low} having a certain value is as follows:

$$\Pr(Z_{low} = n) = \Pr(LB_{up,n} \geq LB_{low,n} \wedge LB_{up,n+1} < LB_{low,n+1}) \quad (10)$$

To approximate the expected value of Z_{low}, we use the same algorithm as in the case of constant functions, for Z. The algorithm calculates and sums up the addends of the following formula one by one as long as these are bigger than a predefined threshold δ:

$$\mathbb{E}[Z_{low}] = \sum_{i \geq 1} \Pr(Z_{low} > i) = \sum_{i \geq 1} \Pr(LB_{up,n} \geq LB_{low,n}) \quad (11)$$

On average, we expect a straight line to approximate a number of points in the order of magnitude of 1. Therefore, here as well, we set $\delta = 0.001$.

Upper Bound To obtain an upper bound for the average length of the segments, we use a model which can approximate at least as many points any straight-line function can approximate within the given error bound. In contrast to our lower-bound model, this model is a relaxation. Here, we do not fix the first point of a possible approximating straight line. Instead, we focus on two approximating lines which pass through the first two points with the highest and, respectively, lowest y-value an approximating line could pass through: $(0,X_0-\varepsilon)$ and $(0,X_0+\varepsilon)$. In other words, we look at a superset of the set of possible approximating lines to establish an upper bound.

We use two sets of random variables: $UB_{up,i}$ and $UB_{low,i}$, with $i \geq 1$. At step i, given the current sequence of samples $[X_0,\ldots,X_i]$, $UB_{up,i}$ is the smallest slope of all straight lines passing through the point $(0,X_0-\varepsilon)$ and one of the points $(1,X_1+\varepsilon)$, $\ldots$, $(i,X_i+\varepsilon)$. Similarly, $UB_{low,i}$ is the largest slope of all straight lines passing through the point $(0,X_0+\varepsilon)$ and one of the points $(1,X_1-\varepsilon),\ldots,(i,X_i-\varepsilon)$. Figure 4 illustrates this for the first three samples X_0, X_1 and X_2. Formally, we calculate $UB_{up,i}$ and $UB_{low,i}$ using the following recursive formulas for $i \geq 2$:

$$UB_{up,i} = min(UB_{up,i-1}, \frac{X_i + 2\cdot\varepsilon}{i}) \quad (12)$$

$$UB_{low,i} = max(UB_{low,-1}, \frac{X_i - 2\cdot\varepsilon}{i}) \quad (13)$$

with

$$UB_{up,1} = X_1 + 2\cdot\varepsilon$$
$$UB_{low,1} = X_1 - 2\cdot\varepsilon$$

The necessary and sufficient condition for our upper-bound model to be able to approximate the sequence of samples $[X_0,\ldots,X_i]$ within the predefined error bound ε is:

$$UB_{up,i} \geq UB_{low,i} \quad (14)$$

If Condition 14 is not fulfilled, this construction of an approximation fails. The following lemma, whose proof is in the appendix, shows that our model is an upper bound for the average length of the segments:

Lemma 3 *Let a sequence of data points* $[(t_0,X_0),\ (t_1,X_1),\ \ldots,\ (t_m,X_m)]$ *be given. If there is a straight line that approximates this sequence of points with a given error bound* ε *then* $UB_{up,m} \geq UB_{low,m}$.

Next, we determine the expected value of the number of consecutive points our upper-bound model can approximate – the random variable Z_{up}. The probability of Z_{up} having a certain value is as follows:

$$\Pr(Z_{up} = n) = \Pr(UB_{up,n} \geq UB_{low,n} \wedge \tag{15}$$

$$UB_{up,n+1} < UB_{low,n+1}) \tag{16}$$

We then use the same algorithm as in the previous cases to calculate the expected value of Z_{up}. Our algorithm calculates and sums up the addends of the following formula one by one, as long as these are bigger than a predefined threshold δ:

$$\mathbb{E}[Z_{up}] = \sum_{i \geq 1} \Pr(Z_{up} > i) = \sum_{i \geq 1} \Pr(UB_{up,n} \geq UB_{low,n}) \tag{17}$$

As before, we set $\delta = 0.001$. Logically, a lower value for δ improves our approximation for the upper bound, as the algorithm sums up more addends of the last formula. However, as we calculate an upper bound, not having the exact result is not detrimental. Leaving out the last addends reduces the upper bound and thus lets it become closer to the real average length of the segments.

4.2 Generation-Based Estimation

In the following, we first present the intuition behind the generation-based estimation method. We then describe its main algorithm.

4.2.1 Intuition

As for the previous method, we rely on the distribution D of the differences between consecutive values of a time series. We assume that data generation is equivalent to generating i.i.d. samples D_1, D_2, ..., D_n of D. Using this assumption, we generate a sufficiently large number of time series, each of which is compressible in one piece (i.e., using one function) within the given error bound. The average length of these time series is an estimation of the average length of a segment.

4.2.2 Algorithm

To efficiently determine the length of the time series we generate, we use the method originally described in [6]. It determines the minimum number of segments necessary for a piecewise approximation of a time series with an error bound using a given set of functions, e.g., polynomials of a fixed degree p. We describe our algorithm (Algorithm 3) in the following.

We generate N time series using D (Lines 6–17). We use the Law of Large Numbers to determine N, such that we obtain accurate estimates (precision error $< 5\%$) with a 95% confidence interval. The algorithm first initializes a list of N time series using samples from D (Line 8). For each time series *ts*, our algorithm generates and adds $2,4,\ldots,2^j$ ($j \geq 1$) points to *ts* and approximates *ts* at each step by the corresponding function (Lines 11–13). We use the same algorithms for approximation as in our compression technique. The algorithm adds points as long as the error bound given by ε is guaranteed (Line 10). The algorithm then performs a binary search on the interval given by the last $2^{(j-1)}$ points (Line 15). It does so in order to find the time series of maximum length whose approximation with the given function is within ε. The algorithm then adds *ts* to the list of "complete" time series (Line 17). It then uses this to estimate the average length (Line 18).

Algorithm 3 Generation-based estimation

```
 1: Let ε be the predefined error bound
 2: Let estimate = 0
 3: Let N be the number of generated time series
 4: Let D be the distribution of the differences
 5: Let time_series_complete = {}
 6: for i = 1 : N do
 7:     j = 1
 8:     current_segment = initialize_segment(D)
 9:     current_error = approximate(current_segment)
10:     while current_error < ε do
11:         current_seg = add_next_points(D, j)
12:         current_error = approximate(current_segment)
13:         j = 2 · j
14:     end while
15:     l = perform_binary_search(current_seg, j)
16:     time_series_complete.add(ts)
17: end for
18: estimate = get_average_length(time_series_complete, N)
```

4.3 Evaluation of Runtime Performance

We evaluate the accuracy of the methods in Section 5. In the following we describe and compare their runtimes.

The runtime of the model-based estimation depends on the number of iterations it executes before the addend it calculates at each step becomes smaller than δ. In contrast to this, the runtime of the generation-based method depends on the regression technique used. For constant functions, it obtains an estimate in $\mathcal{O}(N \cdot l)$, where l is the average length of the time series generated. In the case of polynomials of degree $p \in \mathbb{N}^+$, it runs in $\mathcal{O}(N \cdot p \cdot l \cdot log(l))$.

Table 3 shows the runtimes of both methods for the case of constant functions for one random time series of the REDD dataset (see Section 5.1). We obtained similar results for all other time series we have tested. For the model-based estimation, we have used 1% of the length of each time series as sample size. For the generation-based estimation, we set N to $10,000$. This allows us to obtain estimates with an error precision of less than 5% and a confidence interval of 95%. The results show that the generation-based method takes around 2 times more time.

Table 3 Runtime Comparison – Constant Functions (seconds)

estimation type	Maximum deviation allowed (Ws)					
	1	2	5	10	25	50
model-based	0.46	0.50	1.07	2.15	6.48	9.70
generation-based	0.66	0.80	2.16	4.91	15.68	23.47

Table 4 shows the runtimes of both methods for the case of straight-line functions for one random time series of the Smart dataset. We obtained similar results for all other time series we tested. In this case, one result is that the generation-based method takes on average 35 times more time.

Table 4 Runtime Comparison – Straight-line Functions (seconds)

estimation type	Maximum deviation allowed (Ws)					
	1	2	5	10	25	50
model-based	2.3	4.7	12.3	22.1	43.1	69.8
generation-based	39.9	106.1	353.13	856.2	2190.4	3753.2

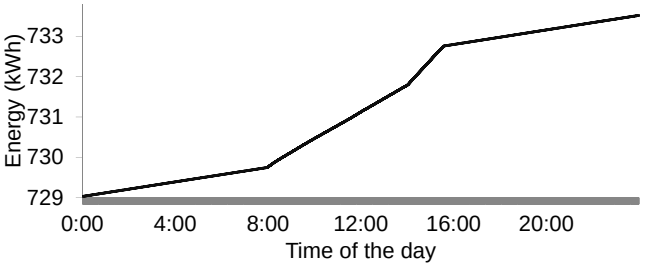
5 Evaluation

In this section, we describe our experimental evaluation. At first, we present the datasets we use (Section 5.1). We then introduce three scenarios for the evaluation (Section 5.2). After that, we present and discuss the results (Section 5.3). We then discuss and evaluate one important aspect of our compression technique, the approximation of additional points (Section 5.4). Finally, in Section 5.5 we evaluate our methods for estimating compression performance.

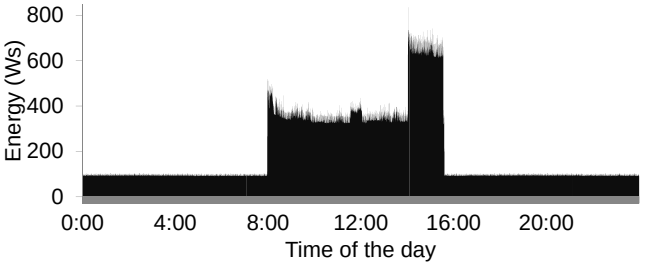
Later in this article, in Sections 6 and 7, we present additional experimental results including a comparison with an alternative approach and results of our technique implemented in a database-management system.

5.1 Datasets for Experimentation

We now describe the datasets used in our evaluation.



(a) *Cumulative representation* – each point refers to the total consumption up to the respective point in time.



(b) *Standard representation* derived from Figure 5(a) – each point refers to the energy consumed in the interval between the previous and the current point.

Figure 5 Two different data representations (one day, office dataset).

5.1.1 Office Dataset

This data is from a smart meter installed in an office of two people [42]. Measurements are done every second. Each measurement contains the amount of energy consumed up to the moment in time when the measurement was taken. This is, the values measured are monotonically increasing. By subtracting adjacent values, this *cumulative representation* of time-series data can be converted into the more common *standard representation* containing the consumption in each individual interval (see Figure 5). We use the cumulative representation for the price-estimation scenario (Section 5.2.1) and the standard representation in the other two scenarios (Sections 5.2.2 and 5.2.3).

5.1.2 The Reference Energy Disaggregation Dataset (REDD)

The REDD was assembled by researchers in the field of energy disaggregation [21] and makes measurements of smart meters in several buildings publicly available. We use data measured second-wise from four individual buildings separately for our experiments. See Figure 5.3.2 for an example of the energy consumption in the REDD in standard representation.

5.1.3 Smart Home Data Set*

This is the Smart* Home collection of datasets [2], which we refer to as *Smart** throughout this paper. It is a public

collection of datasets which researchers in the field of optimizing home-energy consumption have assembled. We use data measured second-wise from two individual houses for our experiments.

5.1.4 Campus Dataset

Besides the three rather fine-grained office, REDD and Smart* datasets, the campus dataset represents data at a level of aggregation higher than the two other datasets, in two aspects: (1) it represents measurements every 15 minutes, and (2) it does not refer to a single office or household, but to the consumption of an industrial campus consisting of four large office buildings with a total of about 4,000 workplaces.

5.2 Evaluation Scenarios

We now introduce three scenarios and derive respective parameters for the compression algorithm. One goal of our evaluation is to identify the highest compression ratio for each scenario which can be reached with the parameters chosen.

5.2.1 Price-Estimation Scenario

Giving consumers access to their consumption data measured by smart electricity meters can result in increased energy efficiency [26]. Our scenario assumes that energy prices are dynamic, i.e., electricity prices which are different at different points in time [8]. This scenario envisions a tool where consumers can browse their energy consumption and costs. Concretely, the user can query the energy costs for a certain time interval.

In a four-person household in Germany, the electricity costs within 15 minutes in a peak hour are equivalent to 0.05 €, according to a standard load profile [27]. We believe that consumers would not be interested in querying time intervals referring to smaller costs. The average minimum energy consumption during 15 minutes is roughly 72 Wh for a four-person household, according to [27]. We choose the maximum tolerable deviation to be less than 5%, i.e., $\pm$1.5 Wh. For our experiments we use the office dataset and the REDD in the cumulative representation as this allows for an easy calculation of energy consumption. Assuming an electricity price of 0.25 €/kWh, the maximum-deviation parameter chosen refers to less than $\pm$0.001 € for an interval in our datasets, i.e., customers will not detect an error in their bill since the deviation does not exceed one cent.

5.2.2 Visualization Scenario

The visualization scenario also contributes to the goal of making users aware of their energy consumption. It makes visualizations of data measured secondly available to individuals. This allows them to visually identify individual consumers such as a microwave. This is not possible when showing only highly aggregated values.

We choose the maximum tolerable deviation to be 25 Ws, $\approx 0.5\%$ of the peak consumption in the REDD or $\approx 3.5\%$ in the office dataset. We choose such a small value to demonstrate that it is possible to compress data measured every second without losing its main advantage, its high precision. For further comparison purposes, we refer to Table 5, which shows the typical power consumption of several standard household appliances [28]. As we can observe, 25 Ws is significantly smaller than the power consumption of most standard home appliances.

Table 5 Example power consumption of standard household appliances.

appliance	power consumption range (W)
coffee maker	900 – 1.200
hair dryer	1.200 – 1.875
microwave oven	750 – 1.100
toaster	800 – 1.400
desktop computer	$\approx$ 270
dishwasher	1.200 – 2.400

5.2.3 Energy Forecast Scenario

Forecasting is a key technique in the smart grid (see Section 2.3). In this article, we do not investigate forecasting as such, but we investigate the effects of our data compression technique on forecasting. We use an out-of-the-box triple-exponential-smoothing algorithm in R to make forecasts on compressed variants of the campus dataset in standard representation. This is a typical source of data for such purposes. For our study, we only investigate a forecast horizon of one day, as this is a standard value in energy consumption forecasting [7].

For our experiments, we chose the maximum tolerable deviation to be smaller than or equal to 250 Wh ($\approx$ 15% of the average energy consumption). We chose this larger value since the campus dataset describes much larger consumptions than the other datasets.

In order to quantify the accuracy of a forecast, we use a set of commonly used forecasting accuracy metrics [7]:

- The *Mean Squared Error (MSE)* is the sum of the squares of the differences between the actual and the predicted values (errors):

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

where y_i correspond to the actual values and y_i' to the predicted ones ($i = 1, \ldots, n$).

– The *Mean Absolute Error (MAE)* measures the sum of the absolute values of the differences between the actual and the predicted values:

$$MAE = \sum_{i=1}^{n} |y_i - y'_i|$$

where y_i correspond to the actual values and y'_i to the predicted ones ($i = 1, \ldots, n$).

– The *Symmetric Mean Absolute Percentage Error (SMAPE)* expresses the accuracy using a percentage, providing both an upper and lower bound on its values. The absolute values of the errors divided through half of the sum of the actual and predicted values are summed and finally divided through the total number of points:

$$SMAPE = \frac{1}{n} \sum_{i=1}^{n} \frac{|y_i - y'_i|}{\frac{1}{2}(y_i + y'_i)}$$

– The *Mean Absolute Scaled Error (MASE)*, proposed as forecasting accuracy metric in [14], is given by:

$$MASE = \frac{1}{n} \sum_{t=1}^{n} \frac{|y_t - y'_t|}{\frac{1}{n-1} \sum_{i=2}^{n} |y_i - y_{i-1}|}$$

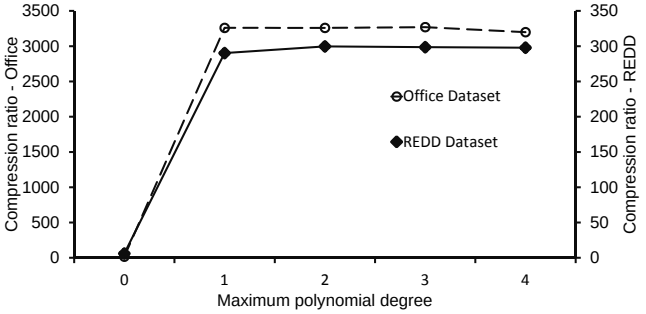
The *MASE* is based on the *MAE*, and it is scaled based on the in-sample *MAE* from the "random walk" forecast method.

5.3 Experimental Results

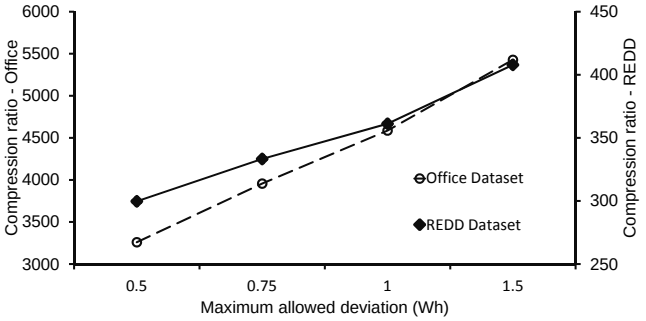
We now present our experimental results in the three scenarios separately, and we compare our approach to related work. All experiments in this section have been executed on a Windows 7 machine using an Intel Core 2 Duo CPU at 3 GHz with 4 GB of RAM. Regarding the compression ratios, we note that we use the function described in Section 3.4 in this section. In realistic deployments of the compression technique, where exact storage requirements are known, the values for compression ratios might differ. We present some further results on a deployment in a database management system in Section 7.

5.3.1 Price-Estimation Scenario

At first we investigate the variation of the compression ratio depending on the maximum degree of the polynomials (p) used within our compression algorithm. Figure 6(a) represents this variation with a maximum allowed deviation of 0.5 Wh. Using only polynomials of degree zero (constant-value functions) on the REDD, the compression ratio is relatively low – around 6. The ratio increases significantly when including polynomials of first degree (straight-line functions). Including second-degree polynomials and polynomials of higher degree increases the compression ratio only moderately. These observations are similar on the office dataset, but on a higher level. Compared to a maximum compression ratio of around 300 on the REDD, the compression ratio is higher than 3,000. This is caused by the nature of this dataset with only a few devices. This leads to an absolute variation of measurement values which is a lot smaller than on the REDD. This is beneficial if the maximum-deviation parameter is – as in this case – chosen in a way that a very small variation can be smoothed by the compression algorithm.



(a) Compression ratio vs. max. degree (p).



(b) Compression ratio vs. max. allowed deviation.

Figure 6 Compression results, price-estimation scenario.

It is interesting to note that the compression ratio can also slightly decrease when including polynomials of degrees higher than three. This is because the algorithm uses a greedy approach. Consider the following example result of our compression algorithm: There are two consecutive intervals, the first one with a polynomial of a high degree, the second one with a polynomial of a low degree. In certain situations, shortening the first interval and starting the second one earlier results in a higher compression ratio. For instance, this is the case if the extended second interval can be compressed with a polynomial of the same degree as before, while the shortened first interval can be compressed with a polynomial of a lower degree than before (i.e., fewer parameters and thus better compression). Our compression algorithm might however not find such rare situations, caused by its greedy approach. This does not guarantee optimal results, but yields good results in relatively low runtime, according to the results presented in this section.

We now consider the variation of the compression ratio depending on the value of the maximum deviation allowed. Figure 6(b) shows how the maximum compression ratios achieved vary for different maximum deviations allowed (up to second-degree polynomials). The curves show that the compression ratios grow linearly depending on the maximum deviation allowed. Again, the observations are similar for both datasets.

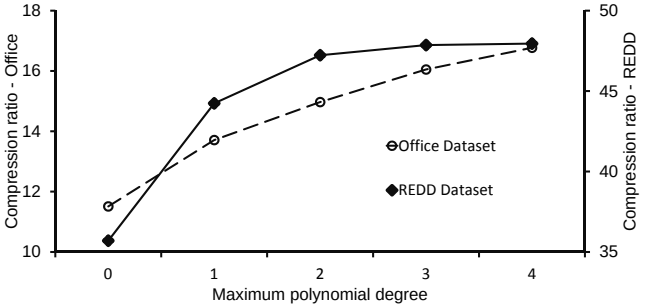
To better understand how our compression algorithm works with the different datasets, we give some information regarding the distribution of the different functions employed by our algorithm (polynomials up to the fourth degree). The most frequently used polynomial is the straight-line function with around 72% of the cases with the REDD (83% to 91% in the office dataset). The polynomials of the second degree occur in about 12% of the cases with the REDD (9% to 16% in the office dataset), while those of the third degree in about 10% with the REDD. Polynomials of the fourth degree are employed in about 5% of the cases with the REDD, while the least frequently used polynomials are those of degree zero, which are never used in both datasets. This is because the data used in this scenario is in the cumulative representation. Thus, constant-value functions are not adequate to fit longer segments of points, while straight-line functions can compress the data well. In the office dataset, polynomials of the third and fourth degrees are never used either. This can be explained by the consumption behavior with less variation of this rather fine-grained dataset compared to the coarser REDD: Simpler functions, i.e., the straight-line function and the parabolic function, suffice to compress the dataset.

Besides the compression ratios and the distribution of regression functions used, the runtime is another important aspect of our evaluation. We measure it for different values of the maximum deviation allowed and the maximum polynomial degree. Table 6 presents the average runtimes of the compression algorithm on the REDD.

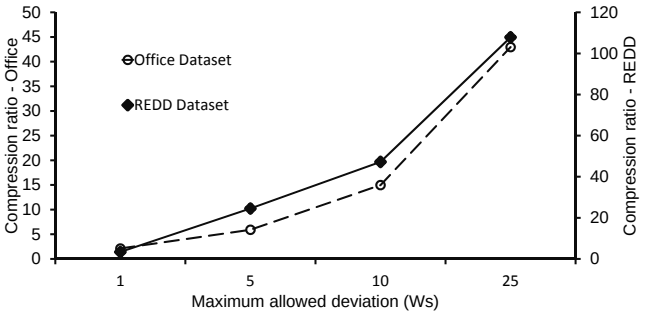
Table 6 Average runtime (seconds) per day (REDD; price estimation).

	maximum deviation allowed (Wh)			
max. degree (p)	0.5	0.75	1	1.5
0	5.28	4.50	4.18	3.82
1	30.40	29.83	32.53	31.74
2	30.05	29.50	31.84	32.36
3	30.80	29.69	30.86	32.01
4	29.96	29.46	30.51	32.61

When using only polynomials of degree zero, the algorithm proves to perform with the smallest runtime on both datasets. In this case, the average runtime corresponds to about four to five seconds on both datasets, decreasing with a growing value for the maximum deviation allowed. Including polynomials of the first degree increases the runtime up to an average of around 30 seconds on the REDD (84 to 113 seconds on the office dataset). Interestingly, when polynomials of higher degree are included as well, the runtimes do not change significantly, remaining at a constant average of about 30 seconds on the REDD (87 to 115 seconds on the office dataset). The explanation of this surprising result is an effect of our implementation using R and various libraries: The linear programming algorithm that outputs the approximation using polynomials of degree $k \geq 2$ is implemented in C and is used within R with the help of a wrapper. Programs in C however are known to perform several times faster than those in R. Thus, although results should have shown longer runtimes for the inclusion of polynomials of larger degree, we did obtain equivalent runtimes because of this implementation detail.



(a) Compression ratio vs. max. degree (p).



(b) Compression ratio vs. max. allowed deviation.

Figure 7 Compression results, visualization scenario.

5.3.2 Visualization Scenario

We again firstly present the results describing the compression ratio depending on various maximum degrees (p) of the polynomial functions. Figure 7(a) shows the compression ratio with a maximum deviation allowed of 10 Ws. It shows that including polynomials of higher degrees always increases the compression ratio in both datasets. Compared to the price-estimation scenario, polynomials of higher degree prove to be a lot more useful. This is due to the non-existing effect of the cumulative representation in this scenario.

Figure 8 illustrates the compressed and decompressed versions of a time series of the REDD. In this case, the maximum deviation allowed is of 25 Ws, corresponding to a compression ratio of factor 108 (see Figure 7(b)). We can observe that with our highest deviation tolerable, apart from a smoothing effect, it is difficult to visually notice any significant differences between the two versions. Moreover, it is possible to visually identify individual devices, which is helpful for energy-consuming households to detect and eliminate devices with high power consumption.

Figure 7(b) (using polynomials up to degree four) shows for both datasets that the compression ratio grows almost linearly depending on the maximum deviation allowed, as in the price-estimation scenario. The compression ratio grows from 4 to 108 with the REDD (3 to 47 with the office dataset).

In general, it is notable in this scenario that compression ratios are lower with the office dataset than with the REDD. However, the situation has been the other way round in the price-estimation scenario. This can be explained by the different nature of the two datasets and the different choice of parameters in the two scenarios: While the more fine-grained office dataset was better compressible in the price-estimation scenario where small variations in the curve could be smoothed by compression, small variations now need to be kept in order to allow for a fine-grained visualization.

5.3.3 Energy Forecast Scenario

The energy-forecast scenario investigates the capability to do forecasts on compressed data. To this end, we have firstly compressed the campus dataset with various maximum deviations allowed, using polynomial functions up to the fifth degree. Preliminary experiments have revealed that the compression ratio does not vary significantly if polynomials of higher degrees are included. We have used a cleaned version of the dataset without days with unusual consumption behavior. The rationale has been to focus on the influence of the compression rather than observing varying forecast qualities.

We have calculated the forecast using both the uncompressed and the compressed versions of the dataset. We have used a common evaluation strategy, namely using a part of the data for the estimation of the forecasting model parameters (training data) and another part for measuring forecasting accuracy (test data) within a sliding-window approach. In each step of this approach (consisting of around 8,000 steps in total), we have calculated the forecast. During this process, we have derived error metrics (see Section 5.2.3) in comparison to the original uncompressed data and have averaged them at the end.

Figure 9 shows the values of the different error metrics on the campus dataset compressed with different values for the maximum deviation allowed. The column corresponding to a maximum deviation allowed of 0 Wh shows the results for the original data – it is our baseline. Overall, the results are as follows: For maximum allowed deviations of less than 25 Wh, the results of the forecasting algorithm do not vary significantly when compared to results for the actual data for three out of four error metrics. At the same time, the value for these error metrics can be even slightly smaller than for the actual data. This is caused by a smoothing effect of the data induced by our compression algorithm: The regression functions used for compression smoothen the curve and eliminate small variations. This makes it easier for the forecasting function to predict the future development of the curve in some situations. For maximum deviations allowed bigger than 50 Wh, the value for *MASE* is significantly larger than the value for the actual data.

Table 7 Compression ratios on the campus dataset (forecast).

max. deviation	10Wh	25Wh	50Wh	100Wh	250Wh
compr. ratio	2.05	3.11	5.14	10.63	34.00

Table 7 shows the values of the compression ratios obtained for the different values of the maximum deviation allowed, corresponding to the error metrics in Figure 9. Thus, when guaranteeing a maximum deviation of 25 Wh – which corresponds to a compression factor of 3 – the forecast precision remains unaffected. For three out of the four error metrics, data compressed with factor 11 does not affect the results negatively, and only data compressed with factor 34 affects the results significantly. Thus, depending on the error metric relevant for the forecast application, the data compressed with factor 11 and factor 34 may even be useful.

5.3.4 Comparison to Related Work

To compare our approach to related work, we compare our compression ratios (as discussed in the previous subsections) to the individual regression functions: constant-value functions [22] ('constants' in Table 8) and straight-line functions [9] ('lines' in Table 8). We do not compare our compression technique to techniques based on Fourier [10] or wavelet [5] decomposition. As mentioned in Section 2.2, these techniques do not provide any guarantee for the compressed version of the data.

Table 8 contains the compression ratios of all three approaches in all three scenarios on the different datasets. We set the maximum degree of polynomials (p) to 3. In the price-estimation scenario, it is obvious that constant-value functions cannot compress the data well, as it is in the cumulative representation where the data is typically not constant. Constant-value functions are better suited in the other two scenarios (where the standard data representation is used),

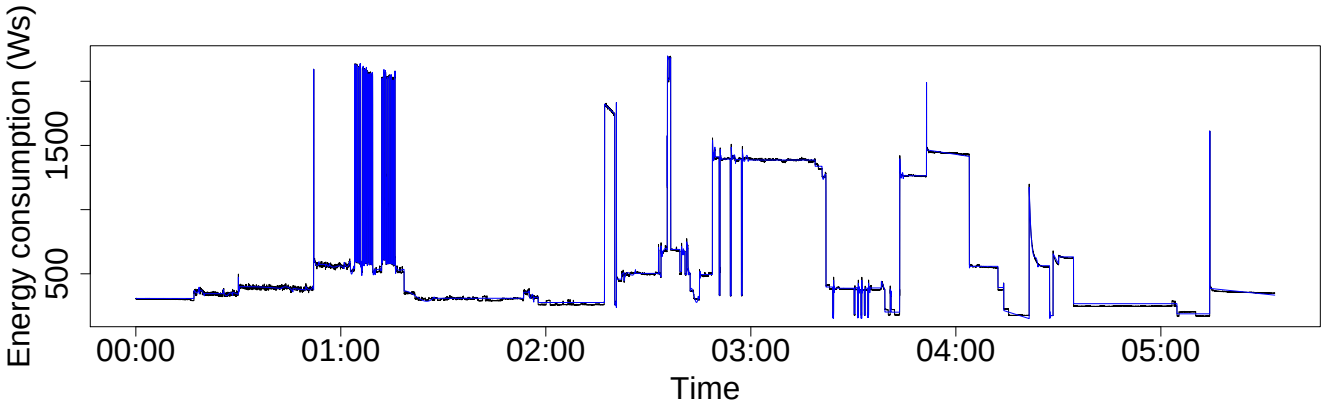
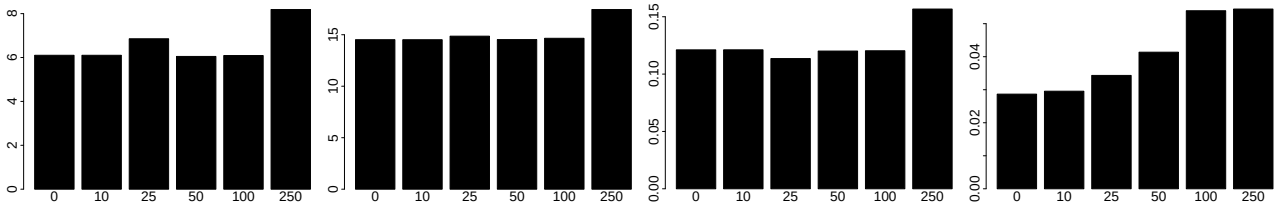


Figure 8 Visual comparison of two curves corresponding to original (black) and compressed (blue) data of the REDD.



(a) *MSE* (MWh^2) (b) *MAE* (kWh) (c) *SMAPE* (d) *MASE*

Figure 9 Energy forecast results on compressed data using four error metrics (the horizontal axis depicts the maximum allowed deviation in Wh).

Table 8 Compression ratios compared to related work.

(a) Price-estimation scenario.

technique & dataset		0.5Wh	0.75Wh	1Wh	1.5Wh	avg.
constants	REDD	6.1	8.3	10.8	14.8	10.0
lines	REDD	290.2	325.0	349.4	391.5	339.0
our approach	REDD	299.7	333.3	361.2	407.9	350.5
constants	office	16.0	23.9	31.7	47.3	29.7
lines	office	3,260.0	4,018.1	4,669.7	5,399.3	4,336.8
our approach	office	3,259.6	3,956.8	4,587.0	5,427.6	4,307.7

(b) Visualization scenario.

technique & dataset		1Ws	5Ws	10Ws	25Ws	avg.
constants	REDD	2.36	17.30	35.70	96.56	37.98
lines	REDD	2.87	21.16	42.16	102.09	42.07
our approach	REDD	3.37	24.53	47.23	107.88	45.75
constants	office	1.69	4.57	11.51	33.08	12.71
lines	office	1.89	5.36	13.52	39.01	14.95
our approach	office	2.42	6.61	16.77	46.94	18.19

(c) Energy forecast scenario (campus dataset).

technique	10Wh	25Wh	50Wh	100Wh	250Wh	avg.
constants	1.14	1.55	2.31	3.98	9.88	3.77
lines	1.43	2.23	3.68	7.69	18.56	6.72
our approach	2.05	3.11	5.14	10.63	34.00	10.99

but their compression ratio is always worse than straight-line functions and our approach. The constant-value functions perform always worse than straight-line functions and our approach. This also holds for the other scenarios and datasets. Comparing our approach to the straight-line functions in Table 8(c) reveals that our approach always compresses the campus dataset better, too. In the price-estimation scenario using the office dataset however, we have observed a few situations where our approach has performed slightly worse. As discussed before, this dataset can be particularly well compressed using straight lines due to the cumulative representation of the data and few changes in consumption. In this situation, the greedy behavior of our compression technique leads to slightly worse results. To sum up, our

approach compresses the data by up to 64% better (on averaged maximum-allowed-deviation values) than the best alternative approach (forecast scenario, Table 8(c); factor 10.99 vs. factor 6.72). As it internally falls back to these approaches, it ensures that best compression ratios can be achieved anyhow, except for pathological situations.

Compared to an implementation of [31] (using only regression functions that fulfill the uniform norm), our approach reaches the same compression ratios, but achieves a speed-up factor of up to 3.

Another factor which impacts the compression ratio is the presence of outliers in the data. We have tested our approach with real datasets which do contain outliers. The result is that our technique achieves high compression ratios when the data contains outliers which occur with a natural frequency. Moreover, as our technique in the worst case falls back internally on related approaches, it will function at least as well as related work on data with unusually high rates of outliers.

5.4 Approximation of Additional Points

After having evaluated our compression technique, we now discuss one interesting aspect and present further results.

Our time-series compression technique automatically divides a time series into intervals and describes the values in these intervals with regression functions. These intervals cover all points of a time series. Thus, it is possible to approximate the values of arbitrary points in the time series. In many cases, the retrieved values might be quite close to the real ones. However, our compression algorithm only provides guarantees regarding a maximum deviation for all points that have been used as an input for the algorithm. This is, there is no guarantee for approximating additional points, and severe deviations may exist when doing so. The fact that there are no guarantees for additional points is general in nature, as one cannot provide guarantees for values which have never been measured or seen by the algorithm. From a machine-learning perspective [29], compression of data can be categorized as highly overfitted learning of the data. This is in contrast to non-overfitted regression learning which may be more suited to approximate unseen data points.

In additional experiments, we have evaluated how well energy data can be compressed when not all points of a dataset are used. To this end, we have compressed our three datasets – but only every second, third, fourth etc. point. Then we have compared the approximated values from the compressed data (including points which have not been used for compression) with all points from the real data. Besides a reduced storage need when not considering all points, the results are as follows: For the campus dataset (using a maximum deviation of 50 Wh), the maximum error of using every point is 50 Wh, and the average error is 27 Wh. For using every second point, the maximum error is 803 Wh (average: 35 Wh), and for using every fourth point, the maximum error is 1,593 Wh (average: 50 Wh). For the office dataset and the REDD, this behavior is roughly similar. To summarize, while maximum errors grow quite quickly, the average errors grow relatively moderately.

From the discussions and the experiments in the previous paragraphs, we conclude that our algorithm can technically approximate points which have not been used for compression. Depending to a high degree on the dataset, the *maximum* error can however be quite high. On the other hand, the results on our energy datasets are quite well *on average*. This is, in certain situations, our algorithm may be used to approximate additional points. Leaving out points for compression might even be a means to increase compression ratios: While the average errors only increase moderately, the compression ratio can be raised by roughly 50% by leaving out every second point (according to experiments with the campus dataset and a small maximum deviation of 10 Ws). This effect almost vanishes when maximum deviation thresholds are large (e.g., the compression ratio raises by only 6% at 250 Ws in the campus dataset). However, one has to be aware that there are no guarantees on the error. These can only be provided for points that have actually been used for compression.

5.5 Evaluation of Estimation Methods

In the following we present an evaluation of our estimation methods described in Section 4. We use two measures to quantify their accuracy. The first one is the absolute percentage error (APE):

$$APE = \frac{|y' - y|}{y} \cdot 100\%$$

where y corresponds to the actual average length of the segments and y' to the one that our method has estimated.

The second measure is the accuracy of our methods when predicting the function (constant, straight line, polynomial of degree $p > 1$) to use to compress a time series. To this end, we first use each of our methods to estimate the size of a time series compressed using a given function. In the case of the model-based estimation, we estimate the average length of the segments for the straight-line function as follows:

$$\text{average length} = \frac{(\text{lower bound} + \text{upper bound})}{2}$$

Based on the estimations, we choose the type of function (constant, straight line, polynomial of degree $p \in \{2,3\}$) that yields the smaller size of the compressed time series. We

then calculate the real size of the compressed time series directly by compressing it using each type of function. Finally, we use the real results to check if our choice has been correct. The measure, which we refer to as *decision accuracy* (*DA*) in the following, is equal to:

$$DA = 100\% \cdot \frac{\text{number of correct decisions}}{\text{all decisions}}$$

5.5.1 Model-Based Estimation

We compared the values the method has produced to actual values obtained by compressing the time series using constant and straight-line functions. We have tested our method on two datasets: REDD and Smart*. To obtain the expected values of Z, Z_{low} and Z_{up} we have sampled each first-difference time series of the datasets. We have used 1% of the length of each time series as sample size. Varying this size from 0.1% to 5% has not had any significant effect on the results. As described in Section 4, we have set δ to 0.001. Using smaller values instead has not improved results significantly. We have performed all experiments 10 times with different samples and present average values in the following. Any result from each of the 10 experiments does not deviate by more than 5% from the average.

Constant Functions We first present detailed results for one random time series of each dataset tested. We then present a summary of the results for all time series.

Table 9 shows the actual and estimated average length of the segments for the time series of the energy consumption of house 2 of the REDD dataset. All in all, the estimation is rather accurate. The maximum absolute percentage error (APE) is equal to 32.5% while the average APE is equal to 12.95% for the different values of the maximum deviation that we tested.

Table 9 Constant functions, real and estimated values, REDD house 2.

	Maximum deviation allowed (Ws)					
	1	2	5	10	25	50
actual	20.14	38.38	68.67	94.81	155.73	213.59
estimated	13.60	29.09	61.08	91.68	149.07	208.57
APE (%)	32.50	24.20	11.05	3.30	4.23	2.35

Table 10 shows the actual and the estimated average length of the segments for the time series of home C of the Smart* dataset. These results are consistent with the previous results just described. Here, the maximum APE is equal to 22.02%, and the average APE is equal to 8.52%.

Concerning all the time series in each of the datasets, Table 11 lists the average APE for each value of the maximum deviation allowed ε that we have tested. As we can observe, our model achieves a good accuracy for the lowest and the highest values of the maximum deviation allowed that we tested. For other values of the maximum deviation allowed, the accuracy is worse, but all in all, our model achieves results with a rather good accuracy. The average APE is smaller than 30% for all values of the maximum deviation allowed that we tested.

Table 10 Constant functions, real and estimated values, Smart* home C.

	Maximum deviation allowed (Ws)					
	1	2	5	10	25	50
actual	0.94	2.27	7.48	13.16	21.63	30.63
estimated	1.00	2.14	5.83	11.17	21.45	31.00
APE (%)	6.14	5.78	22.02	15.12	0.86	1.22

Table 11 Constant functions, average APE.

Dataset	Maximum deviation allowed (Ws)					
	1	2	5	10	25	50
REDD	15.79	20.87	30.70	25.26	19.83	9.19
Smart*	4.36	9.15	23.38	17.56	6.67	5.60

We observed that, apart from one case, our method underestimates the real average length of the segments. We believe this is due to two factors. First, differentiating the time-series does not completely eliminate correlation between consecutive values, i.e., the samples are not entirely i.i.d. We believe this is also why our method produces better results for larger values of the maximum allowed deviation. As the maximum allowed deviation grows, longer sequences of variable data can be approximated using constant functions. The impact of correlation between consecutive samples is thus less significant. Second, Algorithm 2 may underestimate the actual value due to δ. One possibility to improve the estimation would be to use a method which fully calculates $\mathbb{E}[Z]$.

Straight-Line Functions As for constant functions, we will first present detailed results for one random time series of each of the datasets. We will then present average results for all the time series in the datasets.

Table 12 shows the actual average length of the segments, as well as the lower and upper bound values our method produced, for the time series of house 3 of the REDD dataset. As expected, the lower and upper bounds are smaller and, respectively, larger than the actual values for all cases. The lower bound is around 7 – 54% smaller than the actual average length of the segments. The upper bound is around 19 – 55% larger.

Table 13 shows the actual average length of the segments, as well as lower and upper bound values, for the energy consumption of home B of the Smart* dataset. Here as well, the lower and upper bounds are smaller and, respectively, larger than the actual values for all values of the max-

Table 12 Straight-line functions, real, lower and upper bound values, REDD house 3.

	Maximum deviation allowed (Ws)					
	1	2	5	10	25	50
lower bound	3.2	7.7	19.8	35.6	72.5	121.9
actual	6.1	12.9	27.8	49.8	102.9	158.0
upper bound	7.6	16.1	35.3	61.8	124.8	232.8

imum deviation allowed that we tested. The lower bound is around 13 – 57% smaller than the actual average length of the segments. The upper bound is around 26 – 59% larger than the actual average length of the segments.

Table 13 Straight-line functions, real, lower and upper bound values, Smart* home B.

	Maximum deviation allowed (Ws)					
	1	2	5	10	25	50
lower bound	5.0	18.5	60.9	121.3	253.5	393.6
actual	11.8	33.6	88.0	159.3	308.4	451.4
upper bound	18.7	47.2	127.3	219.2	387.9	630.3

We present in Table 14 average results we obtained for all the time series in each dataset. The table contains the average APE of both the upper and lower bound for each value of the maximum deviation allowed that we tested. Our model produces rather good lower and upper bounds. All in all, the average APE is smaller than 55% for both lower and upper bounds. Generally, the bounds get closer to the actual value as the maximum deviation allowed increases.

Table 14 Straight-line functions, average accuracy (%) of lower and upper bounds.

		Maximum deviation allowed (Ws)					
		1	2	5	10	25	50
REDD	lower bound	48.4	43.5	34.4	25.2	18.8	12.9
	upper bound	24.4	45.6	35.5	36.4	22.8	32.32
Smart*	lower bound	54.7	49.2	40.3	31.0	21.7	12.0
	upper bound	30.4	23.2	27.7	26.5	23.7	33.9

5.5.2 Generation-Based Estimation

In the following we present an evaluation of the generation-based estimation. We compare the values the method has produced to actual values we obtained by compressing the time series using polynomials of different degrees. As for the model-based estimation, we have tested our model on two datasets: REDD and Smart*. To obtain estimations with a precision error smaller than 5% and a confidence interval of 95%, we have generated $N = 10,000$ time series.

Constant Functions As for the model-based estimation, we present detailed results for one time series of each of the datasets tested. We then present a summary of the results for all the time series in each dataset.

Table 15 shows the actual and estimated average length of the segments for the energy consumption of house 4 of the REDD dataset. Except for maximum allowed deviations of 5 and 10 Ws, the APE is smaller than 30%. All in all, the maximum absolute percentage error (APE) is equal to 50.13%, while the average APE is equal to 28.29% for the different values of the maximum deviation that we tested.

Table 15 Generation-based Estimation – Constant Functions – Real and estimated values – REDD house 4

	Maximum deviation allowed (Ws)					
	1	2	5	10	25	50
actual	2.5	6.85	38.56	91.84	216.69	337.16
estimated	2.26	5.45	19.23	48.8	152.25	293.03
APE (%)	9.54	20.37	50.13	46.89	29.74	13.09

Table 16 shows the actual and the estimated average length of the segments for the time series of home B of the Smart* dataset. These results are consistent with the previous results just described. Here, the maximum APE is equal to 41.38%, and the average APE is equal to 25.37%.

Table 16 Generation-based Estimation – Constant Functions – Real and estimated values – Smart* home B

	Maximum deviation allowed (Ws)					
	1	2	5	10	25	50
actual	6.13	22.3	65.38	127.08	250.27	395.51
estimated	5.27	13.07	39.94	87.03	209.42	355.45
APE (%)	14.0	41.38	38.90	31.52	16.32	10.12

Regarding all time series in each of the datasets, Table 17 lists the average APE for each value of ε, the maximum deviation allowed that we have tested. As with the previous method, generation-based estimation achieves a good accuracy for the lowest and the highest values of the maximum deviation allowed that we tested. For other values of the maximum deviation allowed, the accuracy is worse, but all in all, the method achieves results with a rather good accuracy. The average APE is smaller than 30% for all values of the maximum deviation allowed that we tested.

Table 17 Generation-based Estimation – Constant Functions – average APE

Dataset	Maximum deviation allowed (Ws)					
	1	2	5	10	25	50
REDD	16.98	23.28	34.15	30.28	25.25	15.37
Smart*	9.88	23.07	30.37	24.0	8.82	5.60

All in all, the accuracy of the generation-based estimation is similar to the one of model-based estimation. How-

ever, as pointed out in Section 4.3, generation-based estimation takes around two times longer than model-based estimation.

Straight-Line Functions As for constant functions, we will first present detailed results for one random time series of each of the datasets. We will then present average results for all the time series in both datasets.

Table 18 shows the actual and estimated average length of the segments for the time series of the energy consumption of house 1 of the REDD dataset. The maximum absolute percentage error (APE) is equal to 31.24%, while the average APE is equal to 26.4% for the different values of the maximum deviation that we tested.

Table 18 Straight-Line Functions – Real and estimated values – REDD house 1

	Maximum deviation allowed (Ws)					
	1	2	5	10	25	50
actual	3.99	10.21	43.7	85.26	185.45	235,19
estimated	3.03	7.58	24.27	56.01	148.41	221.85
APE (%)	23.97	23.55	31.24	26.02	17.14	5.1

Table 19 shows the actual and estimated average length of the segments for the time series of the energy consumption of home B of the Smart* dataset. The maximum absolute percentage error (APE) is equal to 42.42% while the average APE is equal to 24.83% for the different values of the maximum deviation that we tested.

Table 19 Straight-Line Functions – Real and estimated values – Smart* home B

	Maximum deviation allowed (Ws)					
	1	2	5	10	25	50
actual	11.77	33.57	87.98	159.34	308.36	451.36
estimated	8.14	19.33	57.61	121.15	267.16	434.03
APE (%)	30.87	42.42	34.52	23.97	13.36	3.83

We present in Table 20 average results we obtained for all the time series in each dataset. The table contains the average APE for each value of the maximum deviation allowed ε that we have tested. We observe that the method performs worse than for constant functions for low values (up to 2 Ws) of the maximum deviation allowed. In contrast, it performs better for higher values.

In conclusion, generation-based estimation in general performs well in the case of straight-line functions. Its average APE is less than 34.15% for all time series and values of the maximum deviation allowed we tested. However, generation-based estimation takes significantly more time than model-based estimation (around 35 times more).

Table 20 Generation-based Estimation – Straight-Line Functions – average APE.

Dataset	Maximum deviation allowed (Ws)					
	1	2	5	10	25	50
REDD	23.97	23.55	31.24	26.02	17.14	5.10
Smart*	30.88	34.07	30.39	22.56	11.94	3.92

Polynomials of degree p>1 We show results for polynomials of degree $p \in \{2,3\}$ in the following. We do this according to the evaluation of our compression technique, as it has shown that polynomials of degrees $p \leq 3$ have the biggest impact on the compression ratio. Table 21 shows average results of the APE for all time series in each dataset. We observe that the accuracy tends to decrease when the degree of the polynomial increases. We believe that this has the same reason as with the model-based estimation: Differentiating the time series does not completely eliminate correlation between consecutive values, i.e., the samples of D are not entirely i.i.d. However, as the degree of the polynomial grows, the correlation left out by our assumption has a bigger impact on the estimation.

Table 21 Generation-based Estimation – Polynomial Functions – Average APE

		Maximum deviation allowed (Ws)					
		1	2	5	10	25	50
p=2	REDD	22.7	25.6	34.0	30.6	20.5	5.7
	Smart*	20.1	31.1	32.6	23.6	7.5	12.9
p=3	REDD	24.6	27.2	36.8	31.8	23.0	22.6
	Smart*	18.0	31.6	33.5	21.2	15.6	20.4

In conclusion, although the accuracy becomes worse with a larger p, results are still accurate. We obtain particularly small APEs for the lowest and largest values of the maximum deviation allowed.

5.5.3 Decision Accuracy

We now present results for decision accuracy (DA) for both methods. We use the obtained estimates to make a choice for each time series tested for six different values of the maximum deviation allowed: 1 Ws, 2 Ws, 5 Ws, 10 Ws, 25 Ws and 50 Ws. For the model-based estimation, the decision to be made is to choose between constant and straight-line functions. For the generation-based estimation, this choice is from among polynomials of degree $p \in \{0,1,2,3\}$. Table 22 shows the *DA* for all time series tested.

Concerning model-based estimation, except for one time series, the method is helpful when choosing the type of function to use for the compression. We obtained the lowest *DA* for the time series of house 4 of the REDD dataset. However, we observed the following fact for this time series: In the cases our model did not help us make the right decision,

the difference of compression ratio between using a constant and a straight-line function was small. In other words, the method was close to the correct choice.

Regarding generation-based estimation, we observe that, apart from two cases, the method is helpful when choosing the type of function to use for the compression.

Table 22 Decision accuracy for all datasets tested

		Decision accuracy (%)	
		model-based	generation-based
REDD	house 1	66.7	50
	house 2	100	83.3
	house 3	100	67.7
	house 4	33.3	100
Smart*	home B	100	50
	home C	83.3	83.3

6 Alternative Compression Using Standard Patterns

So far, our compression technique does not make use of any recurring pattern such as the daily, weekly and yearly season, as present frequently in many energy and other datasets. For example, the load curve of energy consumption of two adjacent weekdays frequently is very similar. As exploiting this behavior seems to be promising, we have used several properties of our datasets in order to statistically reduce the data before applying our compression technique, and we have analyzed this alternative experimentally.

To use the most prominent repeating pattern in our energy datasets, the daily load profile, we have calculated the standard daily pattern by averaging, for each dataset separately, all values occurring at the same time of each day. We have then calculated the differences of each day of the datasets to the daily standard pattern. For the campus dataset, which has a strong difference between weekdays and weekends, we have used only the weekdays to investigate the effect of the daily pattern in particular. In productive implementations, one could use two different kinds of patterns for the different types of days or switch to a weekly pattern. Figure 10 illustrates the standard pattern in the campus dataset and depicts the energy consumption of a random weekday in the same dataset.

We have subsequently compressed the differences between the datasets and their standard daily patterns using our technique, and we have compared the compression ratios to those obtained when compressing the original data. Table 23 presents the maximum compression ratios obtained for all datasets for both the original data and the differences to the standard patterns. For the office dataset and the REDD, the compression ratios obtained using the original data are bigger than the ones obtained using the differences to the standard daily patterns for all values of the maximum tolerable deviations. On average, the compression ratios using the original data are about two times and 2.5 times bigger than the ones obtained when compressing the differences to the daily standard patterns for the office dataset and the REDD, respectively. For the campus dataset, the compression of the differences between the original dataset and the daily standard pattern resulted in slightly better compression ratios for small values of the maximum tolerable deviation. For larger values of the maximum deviation, the compression ratios are considerably bigger when using the differences to the daily pattern instead of the original data. As a consequence, the compression ratios for the differences to the daily pattern are on average about two times bigger than the ones obtained when compressing the original data.

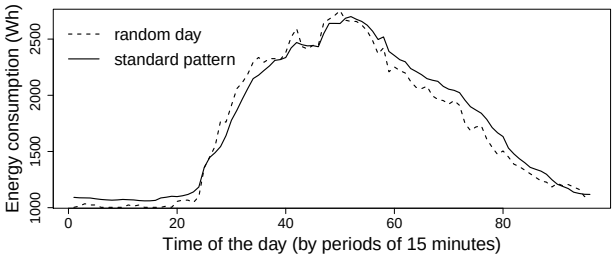


Figure 10 Standard pattern and a random day in the campus dataset.

The low compression ratios achieved using the standard-pattern approach in the office dataset and in the REDD is due to the time granularity of the datasets. As these datasets contain energy consumption measured each second, the differences between the individual days are relatively large. This induces a higher variability of the differences to the daily pattern as calculated above, which accounts for the smaller compressibility of the data. In the campus dataset, the good results are a consequence of the presence of a strong repeating pattern. With this dataset, the energy consumption was measured every 15 minutes, aggregates a large number of consuming devices, and the daily load pattern describes any day of the dataset quite well. This makes the variance of the differences to the daily pattern considerably smaller than the variance of the original data. When the maximum tolerable deviation is large enough, the regression functions are able to approximate longer segments of points, thus making the compression more efficient.

Overall, the standard-pattern approach can lead to better compression ratios in certain situations. However, even if the standard-pattern-based approach can achieve better results in certain situations, it leads to additional overhead for deriving and updating the standard patterns. The approach has therefore not turned out to be a general technique that yields additional benefit in most situations.

Table 23 Original data vs. differences to daily standard pattern: Compression ratios for different max. deviations and their average values.

(a) Office dataset.

data type	1Ws	5Ws	10Ws	25Ws	avg.
original	2.42	6.61	16.77	46.94	18.19
differences	1.53	3.23	6.73	24.26	8.94

(b) REDD.

data type	1Ws	5Ws	10Ws	25Ws	avg.
original	3.37	24.53	47.23	107.88	45.75
differences	3.09	15.06	21.71	31.26	17.78

(c) Campus dataset.

data type	10Wh	25Wh	50Wh	100Wh	250Wh	avg.
original	2.05	3.11	5.16	10.69	34.47	11.09
differences	2.07	3.19	5.55	13.36	81.51	21.14

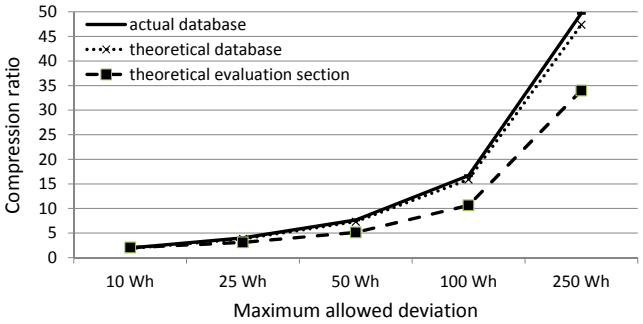
7 Implementation in a Database Management System

While we have conducted all experiments in Section 5 stand-alone with scripts in R, we now discuss how our compression technique can be deployed in a database management system. As mentioned in Section 3.4, compression ratios in reality depend on factors such as storage requirements of individual data types and are hard to predict within a database. Thus, it is important to explicitly evaluate our technique in this very relevant deployment scenario.

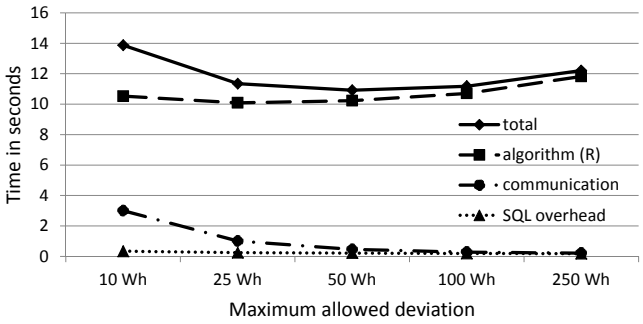
In concrete terms, we use SAP HANA [11], an in-memory relational database management system [32]. To implement our compression algorithm in SAP HANA, we have implemented a function which calls a variation of our R script. This script is executed on an R server running on the same machine as the database engine in our setup. For the implementation, we have used a table schema similar to the one described in Section 3.3 (see Table 1). Knowing the exact storage space requirements of all fields in the actual table scheme allows us to refine the calculation of the compression ratio within the compression algorithm (Line 11 in Algorithm 1, see Section 3.4).

Regarding decompression, we have implemented the respective functionality entirely in the database using pure SQL. This is possible, as – in contrast to our compression algorithm – decompression does not require procedural algorithms or parameter-estimation routines. Using R as an alternative would lead to additional overhead, such as unnecessary data transfers due to the execution in a separated process. For this reason, we have not considered this variant in our evaluation any further. We hypothesize that our SQL-based decompression implementation is efficient.

Figure 11(a) shows the compression ratios of our compression algorithm deployed in SAP HANA on one year of the campus dataset with maximum deviation values as in Section 5.3.3. It includes the compression ratios as calculated by the refined function within our compression algorithm in the database implementation ('theoretical database') as well as the actual compression values as measured with system functions of the database (the actual size of the uncompressed data table divided by the actual size of the compressed data table; 'actual database'). Furthermore, it



(a) Compression ratios.



(b) Runtimes.

Figure 11 Compression on SAP HANA, campus dataset, one year.

includes the compression values of our stand-alone evaluation as given in Table 7 ('theoretical evaluation section'). Compared to these values, the values from our database implementation are quite good, and the value used within the algorithm does not differ from the actual one by much. In the database, we achieve a maximum compression ratio of factor 50 instead of factor 34. This positive effect is caused by the usage of more realistic compression-ratio values than the ones used in our evaluation in Section 5. This also indicates that the compression-ratio results given previously in Section 5 are rather conservative estimates of the compression ratios our technique can reach in real database deployments.

Figure 11(b) shows the runtime of our compression algorithm operating on one year of the campus dataset (≈35,040 measurement points) integrated in SAP HANA running on a shared server. The figure explicitly shows the runtime of the compression algorithm in R ('algorithm (R)'), the overhead of retrieving the uncompressed data and of storing the

compressed data in the database using SQL or SQLScript[1] ('SQL overhead'), the communication overhead needed to transfer data from the database engine to the R server and back ('communication') and the sum of these three values ('total'). The SQL operations consume the smallest part of the runtime and are extremely fast. The by far largest share of the runtime is used for the compression algorithm in R. These values are roughly the same as in stand-alone executions of the compression algorithm as performed in Section 5.3. The corresponding values increase with an increasing maximum deviation, as the algorithm tries to compress the data more in these situations. However, the relatively large runtime requirements are partly caused by the R language used in our implementation. When languages with faster implementations are used to implement the same algorithm, we expect further decreases in runtime. Finally, the communication overhead between the database engine and the R server is significant, for small maximum-deviation values in particular. The reason is the overhead when transferring relatively large volumes of only slightly compressed data between R and the database. We also expect this overhead to become smaller when the algorithm is implemented natively in SAP HANA in languages such as C, C++ or L[2]. Furthermore, we expect the total runtime to decrease further when deployed in a dedicated database system – the results presented here have been obtained on a shared server. This is, the already fast runtimes of our implementation presented here might be further decreased.

Regarding runtimes of decompression, this process is significantly faster than compression, due to its implementation in pure SQL. In concrete terms, decompression is roughly 20 times faster than compression. Decompressing all values from one year of the campus dataset is possible in less than one second and takes only less than two times longer than querying the original uncompressed table.

To summarize the experiments in SAP HANA, we have shown that our compression algorithm can be smoothly integrated in database management systems. This is essential for usage in a productive environment. In our deployment, we can reach even better compression ratios and relatively short runtimes. Besides compression, decompression is possible in a very fast way. Our algorithm could therefore be used as a universal technique as part of the database management system to store time-series data in a compressed way.

8 Conclusions

In smart electricity grids, time-series data of high temporal resolution proliferate, for example from smart electricity meters. This is very similar in many other domains where time-series data is generated. Storing huge amounts of this raw data may be prohibitively expensive in terms of space required, depending on the application scenarios. However, in many cases, this fine-grained data would empower analytical applications such as very-short-term forecasting. This calls for a solution that compresses time-series data so that transmission and storage of fine-grained measurements becomes more efficient. In this article, we have proposed such a technique for time-series data. It compresses data without any significant loss for most existing smart-grid applications. Besides the energy domain, the technique can be applied to time-series data from arbitrary domains. The proposed technique builds on piecewise polynomial regression. Comprehensive experimental results based on three representative smart-grid scenarios and real-world datasets show that our technique achieves good compression ratios. These are comparable to (but compression is faster) or better than related work. In a forecasting scenario, it can compress data without significantly affecting forecast quality. This means that data in higher temporal resolution can be stored using significantly less space than the original "raw" data requires. Further experiments compare our technique to an alternative approach and illustrate the integration into a state-of-the-art database management system. Additionally, we have proposed two methods for estimating the storage space requirements our of compression technique. Our methods are helpful in several ways. First, they can identify the piecewise regression techniques which best compress a given dataset in the context of a given application. Second, our methods can help assessing the benefit of more complex and time-consuming compression techniques.

Potential further research includes: (1) integration with outlier detection (noisy measurement data might be disadvantageous for both, applications and compressibility), (2) investigation how to perform analytical tasks directly on the compressed data and (3) techniques to support value-based queries on the time-series data. Such queries select, say, all time series having values fulfilling certain criteria, without the need of decompressing all time series.

Acknowledgements We thank L. Neumann and D. Kurfiss, who have helped us with the database implementation and respective experiments.

[1] SQLScript is a procedural programming language in SAP HANA.
[2] L is a programming language similar to C used in SAP HANA.

References

1. Aggarwal, S.K., Saini, L.M., Kumar, A.: Electricity Price Forecasting in Deregulated Markets: A Review and Evaluation. International Journal of Electrical Power and Energy Systems **31**(1), 13–22 (2009)
2. Barker, S., Mishra, A., Irwin, D., Cecchet, E., Shenoy, P., Albrecht, J.: Smart*: An Open Data Set and Tools for Enabling Research in Sustainable Homes. In: Workshop on Data Mining Applications in Sustainability (SustKDD) (2012)
3. Box, G.E.P., Jenkins, G.M.: Time Series Analysis: Forecasting and Control. Holden-Day (1976)

4. Brockwell, P.J., Davis, R.A.: Introduction to Time Series and Forecasting. Springer (2002)
5. Chan, K.P., Fu, A.C.: Efficient Time Series Matching by Wavelets. In: Int. Conference on Data Engineering (ICDE), pp. 126–133 (1999)
6. Dalai, M., Leonardi, R.: Approximations of One-Dimensional Digital Signals Under the l^∞ Norm. IEEE Transactions on Signal Processing **54**(8), 3111–3124 (2006)
7. Dannecker, L., Böhm, M., Fischer, U., Rosenthal, F., Hackenbroich, G., Lehner, W.: State-of-the-Art Report on Forecasting – A Survey of Forecast Models for Energy Demand and Supply. Deliverable 4.1, The MIRACLE Consortium, Dresden, Germany (2010)
8. Eichinger, F., Pathmaperuma, D., Vogt, H., Müller, E.: Data Analysis Challenges in the Future Energy Domain. In: T. Yu, N. Chawla, S. Simoff (eds.) Computational Intelligent Data Analysis for Sustainable Development, chap. 7, pp. 181–242. Chapman and Hall/CRC (2013)
9. Elmeleegy, H., Elmagarmid, A.K., Cecchet, E., Aref, W.G., Zwaenepoel, W.: Online Piece-wise Linear Approximation of Numerical Streams with Precision Guarantees. In: Int. Conference on Very Large Data Bases (VLDB), pp. 145–156 (2009)
10. Faloutsos, C., Ranganathan, M., Manolopoulos, Y.: Fast Subsequence Matching in Time-Series Databases. SIGMOD Record **23**(2), 419–429 (1994)
11. Färber, F., May, N., Lehner, W., Große, P., Müller, I., Rauhe, H., Dees, J.: The SAP HANA Database – An Architecture Overview. IEEE Data Engineering Bulletin **35**(1), 28–33 (2012)
12. Feller, W.: The Asymptotic Distribution of the Range of Sums of Independent Random Variables. The Annals of Mathematical Statistics **22**(3), 427–432 (1951)
13. Huffman, D.A.: A Method for the Construction of Minimum-Redundancy Codes. Proceedings of the Institute of Radio Engineers **40**(9), 1098–1101 (1952)
14. Hyndman, R.J., Koehler, A.B.: Another Look at Measures of Forecast Accuracy. International Journal of Forecasting **22**(4), 679–688 (2006)
15. Ilic, D., Karnouskos, S., Goncalves Da Silva, P.: Sensing in Power Distribution Networks via Large Numbers of Smart Meters. In: Conference on Innovative Smart Grid Technologies (ISGT), pp. 1–6 (2012)
16. Karnouskos, S.: Demand Side Management via Prosumer Interactions in a Smart City Energy Marketplace. In: Conference on Innovative Smart Grid Technologies (ISGT), pp. 1–7 (2011)
17. Karnouskos, S., Goncalves Da Silva, P., Ilic, D.: Energy Services for the Smart Grid City. In: Int. Conference on Digital Ecosystem Technologies – Complex Environment Engineering (DEST-CEE), pp. 1–6 (2012)
18. Keogh, E., Chakrabarti, K., Pazzani, M., Mehrotra, S.: Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases. SIGMOD Record **30**(2), 151–162 (2001)
19. Keogh, E., Kasetty, S.: On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration. In: Int. Conference on Knowledge Discovery and Data Mining (KDD), pp. 102–111 (2002)
20. Keogh, E.J., Pazzani, M.J.: An Enhanced Representation of Time Series Which Allows Fast and Accurate Classification, Clustering and Relevance Feedback. In: Int. Conference on Knowledge Discovery and Data Mining (KDD), pp. 239–243 (1998)
21. Kolter, J.Z., Johnson, M.: REDD: A Public Data Set for Energy Disaggregation Research. In: Workshop on Data Mining Applications in Sustainability (SustKDD) (2011)
22. Lazaridis, I., Mehrotra, S.: Capturing Sensor-Generated Time Series with Quality Guarantees. In: Int. Conference on Data Engineering (ICDE), pp. 429–440 (2003)
23. Le Borgne, Y.A., Santini, S., Bontempi, G.: Adaptive Model Selection for Time Series Prediction in Wireless Sensor Networks. Signal Processing **87**, 3010–3020 (2007)
24. Lin, J., Keogh, E., Wei, L., Lonardi, S.: Experiencing SAX: A Novel Symbolic Representation of Time Series. Data Mining and Knowledge Discovery **15**(2), 107–144 (2007)
25. Makridakis, S.G., Wheelwright, S.C., Hyndman, R.J.: Forecasting: Methods and Applications, 3rd edn. Wiley (1998)
26. Mattern, F., Staake, T., Weiss, M.: ICT for Green: How Computers Can Help Us to Conserve Energy. In: Int. Conference on Energy-Efficient Computing and Networking (E-Energy), pp. 1–10 (2010)
27. SWKiel Netz GmbH: VDEW-Lastprofile (2006). URL `http://www.stadtwerke-kiel.de/index.php?id=swkielnetzgmbh__stromnetz__mustervertraege_haendler_rahmenvertrag`. Accessed 25 April 2013
28. U.S. Department of Energy: Estimating Appliance and Home Electronic Energy Use (2013). URL `http://energy.gov/energysaver/articles/estimating-appliance-and-home-electronic-energy-use`. Accessed 20 November 2013
29. Mitchell, T.: Machine Learning. McGraw Hill (1997)
30. Nga, D., See, O., Do Nguyet Quang, C., Chee, L.: Visualization Techniques in Smart Grid. Smart Grid and Renewable Energy **3**(3), 175–185 (2012)
31. Papaioannou, T.G., Riahi, M., Aberer, K.: Towards Online Multi-Model Approximation of Time Series. In: Int. Conference on Mobile Data Management (MDM), pp. 33–38 (2011)
32. Plattner, H., Zeier, A.: In-Memory Data Management – An Inflection Point for Enterprise Applications. Springer (2011)
33. Ramanathan, R., Engle, R., Granger, C.W., Vahid-Araghi, F., Brace, C.: Short-run Forecasts of Electricity Loads and Peaks. International Journal of Forecasting **13**(2), 161–174 (1997)
34. Ratanamahatana, C., Lin, J., Gunopulos, D., Keogh, E., Vlachos, M., Das, G.: Mining Time Series Data. In: O. Maimon, L. Rokach (eds.) Data Mining and Knowledge Discovery Handbook, chap. 56, pp. 1049–1077. Springer (2010)
35. Ringwelski, M., Renner, C., Reinhardt, A., Weigely, A., Turau, V.: The Hitchhiker's Guide to Choosing the Compression Algorithm for Your Smart Meter Data. In: Int. Energy Conference (ENERGYCON), pp. 935–940 (2012)
36. Salomon, D.: A Concise Introduction to Data Compression. Springer (2008)
37. Seidel, R.: Small-Dimensional Linear Programming and Convex Hulls Made Easy. Discrete & Computational Geometry **6**(1), 423–434 (1991)
38. Shahabi, C., Tian, X., Zhao, W.: TSA-tree: A Wavelet-Based Approach to Improve the Efficiency of Multi-Level Surprise and Trend Queries on Time-Serieseries Data. In: Int. Conference on Scientific and Statistical Database Management (SSDBM), pp. 55–68 (2000)
39. Shieh, J., Keogh, E.: *i*SAX: Indexing and Mining Terabyte Sized Time Series. In: Int. Conference on Knowledge Discovery and Data Mining (KDD), pp. 623–631 (2008)
40. Taylor, J.W.: Triple Seasonal Methods for Short-Term Electricity Demand Forecasting. European Journal of Operational Research **204**(1), 139–152 (2010)
41. Tishler, A., Zang, I.: A Min-Max Algorithm for Non-linear Regression Models. Applied Mathematics and Computation **13**(1/2), 95–115 (1983)
42. Vogt, H., Weiss, H., Spiess, P., Karduck, A.P.: Market-Based Prosumer Participation in the Smart Grid. In: Int. Conference on Digital Ecosystems and Technologies (DEST), pp. 592–597 (2010)
43. Wijaya, T.K., Eberle, J., Aberer, K.: Symbolic Representation of Smart Meter Data. In: Workshop on Energy Data Management (EnDM), pp. 242–248 (2013)
44. Yi, B.K., Faloutsos, C.: Fast Time Sequence Indexing for Arbitrary L_p Norms. In: Int. Conference on Very Large Data Bases (VLDB), pp. 385–394 (2000)
45. Ziv, J., Lempel, A.: A Universal Algorithm for Sequential Data Compression. IEEE Transactions on Information Theory **23**(3), 337–343 (1977)

9 Appendix

In the following we present formal proofs for our lower-bound and upper-bound models.

9.1 Proof for Lower Bound

Lemma 2 *Let a sequence of data points* $[(t_0,X_0), (t_1,X_1), \ldots, (t_m,X_m)]$ *and an error bound* ε *be given. If* $LB_{up,m} \geq LB_{low,m}$*, then there is a straight line approximating the sequence of data points within the given error bound.*

The *swing* filter compresses a time series using connected straight lines, while the *slide* filter compresses a time series using disconnected straight lines. To prove Lemma 2, we first show that our lower-bound model is equivalent to the *swing* filter introduced in [9]. We then use the following fact from [9]: If the *swing* filter can approximate a sequence of points under a given error bound, then the *slide* filter can also approximate this sequence under the same error bound.

Proof: At each step i, given a sequence of i data points $[(t_0,X_0), (t_2,X_2), \ldots, (t_i,X_i)]$, the *swing* filter uses two straight lines, u and l. The first line u is the one with the minimum slope of the lines that pass through the first point (t_0,X_0) and one of the points $(t_2,X_2+\varepsilon), \ldots, (t_i,X_i+\varepsilon)$. The second line l is the line with the maximum slope of the lines that pass through the first point (t_0,X_0) and one of the points given by $(t_2,X_2-\varepsilon), \ldots, (t_i,X_i-\varepsilon)$. The definitions of u and l are equivalent to those of $LB_{up,i}$ and $LB_{low,i}$, respectively.

Furthermore, the condition for the *swing* filter to make a recording, i.e., the filter cannot approximate the sequence of points including the newest point (t_{i+1}, X_{i+1}), is as follows:

$$(t_{i+1},X_{i+1}) \textit{ is more than } \varepsilon \textit{ above } u \textit{ or below } l \tag{18}$$

The equivalent condition for our lower bound model (Condition 9) is:

$$LB_{up,i+1} < LB_{low,i+1}$$

Given that $LB_{up,i} \geq LB_{low,i}$ and $LB_{up,i+1} < LB_{low,i+1}$, point $(i+1,X_{i+1})$ falls more than ε above u or below l. This means that the two conditions are equivalent. Thus, our model is equivalent to a *swing* filter. We combine this with the following fact from [9]: if the *swing* filter can approximate a sequence of points under a given error bound, then the *slide* filter can also approximate them under the same error bound. As a result, our model *is* a lower bound for the average length of the segments.

9.2 Proof for Upper Bound

Lemma 3 *Let a sequence of data points* $[(t_0,X_0), (t_1,X_1), \ldots, (t_m,X_m)]$ *be given. If there is a straight line that approximates this sequence of points with a given error bound* ε *then* $UB_{up,m} \geq UB_{low,m}$*.*

Proof: To prove Lemma 3, we use Lemma 4.1 for univariate time series from [9]:

Lemma 4 *Let a sequence of data points* $[(t_1,X_1), (t_2,X_2), \ldots, (t_m,X_m)]$*, such that there exists a straight line that is within* ε *from all the data points be given. If u (l) is a straight line with the following properties:*
(P1) u (l) passes through a pair of points $(t_h,X_h-\varepsilon)$ *and* $(t_l,X_l+\varepsilon)$ *(*$(t_h,X_h+\varepsilon)$ *and* $(t_l,X_l-\varepsilon)$*), such that* $t_1 \leq t_h < t_l \leq t_m$*.*
(P2) u (l) has the minimum (maximum) slope (i.e., dx_i/dt*) among all straight lines fulfilling Property (P1).*
then u (l) also has the following properties:
(P3) u (l) is within ε *from all data points.*
(P4) u (l) has a slope higher (lower) than any other straight line fulfilling Properties (P1) and (P3) for any $t > t_m$*.*

Using Property $(P4)$ from Lemma 4, we conclude that $l \leq u$. By their definition, $UB_{up,i} \geq u$ and $UB_{low,i} \leq l$. Thus, $UB_{up,i} \geq UB_{low,i}$. In consequence, if a straight-line function can approximate a sequence of points within a given error bound, then Condition 14 holds as well.