Lossy Time-Series Transformation Techniques in the Context of the Smart Grid

zur Erlangung des akademischen Grades eines
Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte
Dissertation

von

Pavel Efros

aus Chisinau

Tag der mündlichen Prüfung:	2. Mai 2016
Erster Gutachter:	Prof. Dr.-Ing. Klemens Böhm
Zweiter Gutachter:	Prof. Dr. Hartmut Schmeck

Acknowledgements

This thesis is the result of the contributions of many people. I would like to express my appreciation to everyone who has helped make this possible.

First of all, I would like to express my sincere gratitude to Prof. Klemens Böhm. Throughout the three and a half years at the institute, he helped me with invaluable advice, crucial for the quality of my research. Furthermore, his experienced guidance and tireless editing efforts made it possible to publish my work at top venues.

I would also like to express my deep appreciation to Prof. Hartmut Schmeck. Besides acting as a referee for my thesis, he has taken his time and has offered numerous useful comments that have improved my thesis greatly.

I extend my gratitude to Prof. Erik Buchmann. His expertise, positive insights and valuable research ideas were essential for my work.

I have had the chance to work with many bright students during my time at the institute. I would like to thank them for their valuable insights and assistance in conducting research. Special thanks here goes to Adrian Englhardt.

Our research group has been a constant source of knowledge and inspiration. My colleagues have many times taken the time and the effort to carefully read my papers. Their feedback has been an additional motivation to improve my work. I would therefore like to thank all of my colleagues from the institute. Special thanks go to our secretaries at the chair: Barbara and Bettina. They have always helped me with every administrative issue I was dealing with in a fun way.

I am also particularly grateful to my friends for their constant endorsement and entertainment. The periodic coffee breaks with Yaroslav let me replenish my energy sources and helped me reduce stress and tackle day-to-day research issues better.

I am blessed to have my caring family. They have always been there for me throughout my entire life. My parents and brothers encouraged and enabled me to pursue my education at the institute. I am therefore extremely grateful for their love and encouragement. And last but not least, I would like to thank my wife, Alla, for her unconditional love and support. You helped me bring this thesis to its fortunate end.

Abstract

The goal of this thesis is to address essential challenges emerging from the collection of big volumes of fine-granular time-series data. The first challenge corresponds to the huge amounts of storage space required to save such data. The second challenge involves the big effort often required to acquire sensors to measure and communicate the data. Research has provided promising approaches, in the form of lossy time-series techniques, to deal with these challenges. The major contributions of this work are, on the one hand, the improvement and generalization of such approaches and, on the other hand, the investigation of the impact of such approaches on subsequent data analysis.

In this thesis we develop several techniques and a framework to address the challenges above. The first part proposes a lossy compression method, which approximates time series piecewisely using polynomials of different degrees incrementally. Compared to existing techniques, our method achieves better compression ratios as it can adapt to the data well. Moreover, we investigate the problem of choosing good parameters for our method given a dataset and propose two assisting methods. The second part of our thesis involves another kind of lossy time-series techniques: the estimation of time series instead of their direct measurement with the goal of reducing the effort of data collection. We consider the case of estimating computer energy-consumption, which is an increasing and important share of the total energy consumption. We first analyze and extend existing estimators, then integrate them into a general estimation framework. Our framework is the first to explicitly consider the trade-off between accuracy of estimates and the effort of obtaining them. The third part of our thesis builds on the previous two. Here, we investigate techniques which transform the original data, be it for compression, estimation or other purposes. We analyze the impact of such techniques on subsequent data analyses. Specifically, we consider the case of data-analysis methods based on change detection and propose a general measure for quantifying the impact of lossy transformations on subsequent change detection. Our measure is parameterizable and can be used to identify adequate parameters of a lossy transformation so that its advantages are maximized while the impact on subsequent change detection is bounded.

The thesis includes an extensive experimental analysis based on real-world application scenarios from the Smart Grid domain. We analyze all our techniques using different schemes on real-world energy-consumption datasets. Overall, we believe our work shows that lossy time-series transformation techniques can efficiently deal with the challenges above while maintaining data useful for subsequent analyses.

Deutsche Zusammenfassung

Das Forschungsziel dieser Dissertation ist wesentliche Herausforderungen bei der Erfassung großer Volumen fein-granularer Zeitreihen anzugehen. Die erste besteht darin, dass solche Daten enorme Mengen an Speicherplatz benötigen. Die zweite stellt den großen Aufwand dar, der in vielen Fällen dem Beschaffen von Sensoren zum Messen und Kommunizieren der Daten entspricht. Die jüngste Forschung hat einige vielversprechende Ansätze in Form von verlustbehafteten Technicken vorgeschlagen, um diese Herausforderungen anzugehen. Die wesentlichen wissenschaftlichen Beiträge dieser Arbeit sind auf der einen Seite die Verbesserung und Generalisierung solcher Ansätze und auf der anderen Seite die Untersuchung deren Auswirkungen auf einer nachfolgenden Datenanalyse.

Im Rahmen dieser Dissertation werden einige Techniken und ein Rahmenwerk für das Angehen der oben genannten Herausforderungen entwickelt. Der erste Teil der Arbeit schlägt eine verlustbehaftete Kompressionsmethode vor, die Zeitreihen stückweise mit polynomialen Funktionen von unterschiedlichen Graden inkrementell approximiert. Verglichen mit existierenden Ansätzen, erreicht unsere Methode bessere Kompressionsraten, weil sie sich an den Daten gut anpassen kann. Außerdem untersuchen wir das Problem der Wahl guter Parameter für unsere Methode für einen gegebenen Datensatz und schlagen dafür zwei zusätzlichen Methoden vor. Der zweite Teil dieser Arbeit bezieht sich auf eine andere Art von verlustbehafteten Zeitreihentechniken: die Abschätzung (Approximation) von Zeitreihen anstatt deren direkten Erfassung mit dem Ziel den Aufwand der Datenerhebung zu reduzieren. Wir betrachten im konkreten die Abschätzung des Energieverbrauchs von Rechnern, der einem steigenden und wichtigen Teil des globalen Energieverbrauchs entspricht. Dafür untersuchen und erweitern wir zuerst existierende Schätzer. Danach integrieren wir diese in ein allgemeines Rahmenwerk. Unser Rahmenwerk ist das Erste, das den Trade-off zwischen Genauigkeit der geschätzten Daten und Aufwand der Schätzung explizit betrachtet. Der dritte Teil dieser Arbeit baut sich auf die ersten zwei auf. Wir untersuchen Techniken, die die originalen Daten beispielweise zur Kompression, Abschätzung oder Anonymisierung transformieren. Dabei untersuchen wir die Auswirkungen solcher Techniken auf nachfolgende Datenanalysen. Wir betrachten den Fall von Analysemethoden, die auf einer Erkennung von Änderungen basieren, und schlagen ein allgemeines Maß für die Quantifizierung der Auswirkungen von verlustbehafteten Transformationen auf die nachfolgende Erkennung von Änderungen vor. Unser Maß ist parametrierbar und kann genutzt werden, um angemessene Parameter für eine verlustbehaftete Transformation zu identifizieren. Hierdurch werden die Vorteile der Transformation maximiert und gleichzeitig die Auswirkungen auf eine nachfolgende Erkennung von Änderungen in Grenzen gehalten.

Diese Dissertation enthält eine ausführliche experimentelle Analyse der vorgeschlagenen Methoden mit realen Anwendungsszenarien aus dem Bereich des intelligenten Stromnetzes. Wir analysieren unsere Methoden mit unterschiedlichen Schemata auf realen Energieverbrauchsdatensätzen. Insgesamt schlussfolgern wir, dass verlustbehaftete Transformationstechnicken von Zeitreihen die Herausforderungen gut angehen und gleichzeitig die Nützlichkeit der Daten für nachfolgende Analysen behalten.

Contents

1 Introduction

Time-series data is collected nowadays in many domains [RLG$^+$10] using sensors that measure and communicate values (e.g., temperature, pressure) at ever increasing frequencies. Take for example the Smart Grid, where smart electricity meters measure and communicate data every second or even at higher frequencies. Although such time-series data is useful for many analytical purposes, e.g., monitoring of residential power quality [IKG12], energy trading [Kar11] or visualization [NSQ$^+$12], capturing it involves several challenges: First, collecting time-series at high frequencies requires huge amounts of storage space. As an example, for utility companies serving millions of customers, storing smart-meter measurements of high-resolution would sum up to petabytes of data [EPVM12]. Second, the effort of collection is high since in many cases data must be measured and communicated using sensors, e.g., smart electricity meters. This is expensive in large scales.

Concerning the first challenge above, reducing storage requirements for time series is challenging: On the one hand, decreasing the frequency of capture impacts applications that need high-resolution data, such as energy disaggregation [KJ11]. On the other hand, established lossless compression methods have not been designed for numerical time series and achieve in this case limited success [RRR$^+$12, EEKB15]. To address this problem, recent research has proposed several promising lossy compression methods [EEC$^+$09, LM03]. These achieve good compression ratios by segmenting the time-series and approximating the segments with mathematical functions (piecewise regression, Figure 1.1). However, the performance of such approaches depends strongly on the nature of the data and it is unclear which method to use for a given dataset. Moreover, most approaches use a single class of mathematical functions, e.g., constant or straight-line functions. They, therefore, cannot adapt well to very variable time series.

Regarding the second challenge above, in certain cases, it is possible to obtain time series of estimates instead of direct measurements. As an example, for capturing time series of computer energy-consumption, recent research has provided methods that estimate the energy consumption of computer hardware, instead of measuring it directly [GFS$^+$10, SMPH05]. Such methods are based on technical information [PN11], sophisticated hardware models [NBRS12], or

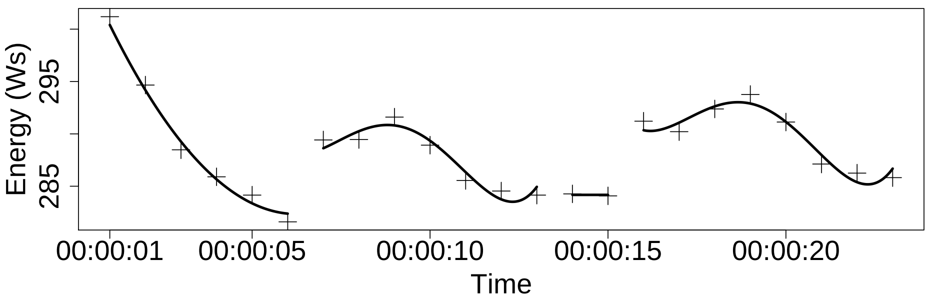


Figure 1.1: Example piecewise compression using regression functions.

profile system and component power usage [RRK08, KZL+10]. However, all of these approaches have different characteristics in terms of setup effort, estimation effort, estimation accuracy and hardware requirements. It is difficult to decide when to use which approach and how to determine meaningful estimation parameters, as well as the accuracy requirements of a given application.

Both classes of methods above (lossy compression or estimation) for addressing the challenges due to the collection of huge volumes of time-series data correspond to lossy approaches, which transform the original data. Although they are promising in reducing storage space or effort of collection, such transformations modify characteristics of the data, which are important for subsequent analyses. In the case of time series, such characteristics may correspond to changes. Change detection on time series data is an important building block of many real-world applications [Pag54, GS99]. It converts a time series of measurements into one of potentially interesting or important events. Consider again energy-consumption data from a smart meter. Change detection on such data allows to detect interesting occurrences (turning on/off of a device, abnormal device activity). Moreover, in the context of the Smart Grid, change detection enables demand side management, peak shifting, peak shaping, etc. – all basic techniques to integrate renewable energy sources into the Smart Grid. However, as mentioned above, data transformation, e.g., lossy compression or estimation, can significantly impact the subsequent detection of those changes. Existing similarity measures for time series, applied to the original series and the compression result, cannot meaningfully quantify the impact of a lossy transformation on the result of an change-detection approach [BC94, RK05, VHGK03, LWKTM07]. Such a quantification however is needed to identify and parametrize a lossy transformation approach, e.g, a compression algorithm, given a certain dataset and quality requirements on the change-detection result.

1.1 Contributions of this Dissertation

The first and second contributions address the challenges related to the collection of large amounts of time series. They include lossy transformation techniques: the first – a compression method and the second – an estimation method. The third contribution builds on the first two. It addresses the challenge of quantifying how such lossy transformation techniques affect the subsequent analysis of the data. In the following, we summarize the contributions of this dissertation:

C1. Storage-Space Reduction Through Adaptive Time-Series Compression: To reduce storage space requirements for time series, we devise a general lossy compression technique for time-series data. 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. Furthermore, 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.

To determine good parameters for our compression technique, we additionally propose two methods for estimating its performance for a given dataset. Our first method ("model-based" estimation), uses a statistical model of the average length of the segments resulting from piecewise regression. It can be instantiated with many piecewise regression techniques. The second method ("generation-based" estimation), although slower than our first method, can be instantiated with any piecewise-regression technique. It efficiently generates a large number of time series with similar characteristics to those in a dataset. It then uses the average length of these time series as an estimate of the average length of segments resulting from the piecewise regression.

C2. Flexible Estimation of Computer Energy Estimation: To determine and reduce the effort of collecting time-series of computer energy consumption, we introduce FRESCO, A FRamework for the Energy eStimation of COmputers. It includes three flexible power-estimation models that generalize state-of-the-art approaches to cover a wide range of accuracy requirements and computer systems. FRESCO integrates these models into an estimation workflow that allows to choose, configure and run an estimator depending on the use case. FRESCO consists of a highly configurable set of estimators, and a workflow to set up and run an instance of an estimator. In particular, FRESCO is able to (a) suggest a set of appropriate estimators for computer energy consumption according to the effort the operator is willing to invest and to the requirements of a certain application, and (b) to execute an instance of the selected estimator with settings that are appropriate for the application. Depending on the kind of estimator, FRESCO can estimate the energy consumption of a computer from various parameters. Such parameters include (1) hardware characteristics, e.g., the energy consumption of a hard disk as specified by its vendor, (2) usage information like CPU load and network activity, and (3) calibration data, e.g., an energy consumption profile that has been recorded by an energy meter for a specific hardware configuration. We evaluate FRESCO with three use cases, namely energy-aware data center management, demand response and energy accounting.

C3. Quantifying the Impact of Lossy Transformations on Subsequent Event Detection: Our previous contributions include techniques which transform time series. To quantify the impact of such lossy transformations on subsequent data analysis, we first investigate application scenarios in detail that involve change detection as subsequent data analysis, which is an important building block of many real-world applications. We then propose MILTON, a "Measure which quantifies the Impact of Lossy Transformation methods on subsequent change detectiON". Our measure is applicable to any use case where one wants to know how much a certain transformation approach for time series reduces the result quality of a change-detection technique, as compared to change detection on the original data. This lets an operator decide how much he can, e.g., compress the data without affecting change detection considerably, or if the perturbation/anonymization technique he intends to deploy does indeed conceal certain changes, as he had planned. To ensure flexibility, we do not impose restrictions on the change detection or the transformation approach used, and we allow to flexibly weight effects on changes. We lastly evaluate MILTON using three Smart Grid application scenarios. Our evaluation shows that it is useful for identifying adequate parameters of a lossy transformation so that its advantages are maximized while the impact on subsequent change detection is bounded.

1.2 Outline of this Dissertation

The remainder of this dissertation presents our contributions in detail:

In Chapter 2 we present related work. We use several real-world datasets to evaluate our contributions. We describe these in Chapter 3.

In Chapter 4 we present our work on lossy time-series compression. We first introduce our adaptive time-series compression method. Subsequently, we show how we evaluated our compression method using three scenarios and three real-world datasets. We then present our methods (“model-based” and “generation-based” estimation) for determining the appropriate time-series compression method and its parameter values for a given dataset. Finally, we evaluate the “model-based” and “generation-based” estimation methods using two real-world datasets.

In Chapter 5 we present FRESCO, our framework for computer energy estimation. We start with a description of application scenarios that cover the spectrum of energy-aware applications for FRESCO. We then explain the classes of effort FRESCO needs to take into account. We subsequently describe the workflow of FRESCO and the estimators it uses. Finally, we present an evaluation of FRESCO using three application scenarios and three real-world datasets.

We start Chapter 6 with the description of three scenarios which motivate MILTON and derives the requirements on it. We then present the functioning of MILTON and ways to parameterize it. We finally evaluate MILTON using three classes of lossy transformations and corresponding real-world datasets. Lastly, in Chapter 7 we summarize the dissertation and present an outlook of open questions and problems which it did not address.

2 Related Work

In the following we describe work related to our approaches and to their application scenarios. We group related work into three categories: time-series data management, computer energy estimation and change detection. Much of this section comes from our publications [EEKB15] (Sections 2.1.1 and 2.1.2), [EBB14a] (Section 2.2) and [EBEB15] (Sections 2.1.3, 2.1.4 and 2.3).

2.1 Time-Series Data Management

Time-series data is of importance in many domains [RLG$^+$10], and it has been investigated for years [BD02, KK02]. Here we first include work related to our first contribution on *time-series compression*. We then consider the topic of *time-series forecasting*, which has gained much attention from the research community [KAMSK09, DBF$^+$10, REWG$^+$97]. As time-series forecasting is very important in smart grids [EPVM12], we use it as one important scenario in the evaluation of our time-series compression technique (Section 4.2). We then consider the topic of *time-series similarity measures*, which is a well-researched area and serves as a basis for our measure presented in Section 6. Finally, we present work related to *time-series anonymization* which we use as application for our measure (Section 6.3.2).

2.1.1 Time-Series Compression

Established compression techniques can be applied to time-series data, which we present first. We next present publications on several data-management and data-mining techniques for time series, such as indexing, clustering and classification [KK02], which provide means for compressed *time-series representation*.

Compression Techniques for Time Series

There are two main categories of compression techniques: *lossless compression* and *lossy compression* [Sal08]. 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. 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*.

The field of research on lossless compression includes established techniques, such as [Huf52, ZL77]. 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 [RRR$^+$12]. 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 [Sal08]. 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 [ZL77]. 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 dissertation 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.

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 [CF99, FRM94, KCPM01, KJP98, LKWL07, STZ00, SK08, WEA13]. 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 [EEC^{+}09, LM03]. 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 [FRM94] to extract features from time series. This helps to build an efficient subsequence-matching algorithm for time series. [CF99] is an approach for a similar time-series-matching problem, but it builds on *discrete wavelet transforms*, particularly *Haar wavelet transforms*. [STZ00] 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 grids.

Another approach for time-series compression is symbolic data representation, which is based on discretization [LKWL07, SK08]. This has recently been applied to smart-meter data [WEA13]. 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” [WEA13] – 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 [HJA13]. 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. [YF00] employs constant functions that approximate fixed-length intervals. [KCPM01, LM03] 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.

[EEC$^+$09] 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. [KJP98] 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 [EEC$^+$09, LM03]) show that these techniques cannot compress highly variable data such as energy consumption data well (see Section 4.2).

The authors in [PRA11] 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, [PRA11] 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, [PRA11] 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.

Model Selection for Time-Series Prediction Another approach for time-series compression in a different setting is presented in [LBSB07]. 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 [PRA11] and to our approach, [LBSB07] employs forecast models which predict future values of a time series instead of regression models. [PRA11] 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 [PRA11] and in contrast to our approach, [LBSB07] 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 [LBSB07] 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.

2.1.2 Time-Series Forecasting

In recent research, an impressive number of forecasting techniques has been developed particularly for energy demand [KAMSK09, DBF$^+$10, REWG$^+$97]. The author in [Tay10] 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 [MCWJH98]. 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.

2.1.3 Time-Series Similarity Measures

Time-series similarity is a well-researched area. The Euclidean Distance is a frequently used distance. Another measure, Dynamic Time Warping (DTW), is commonly used to align sequences [BC94, RK05]. The DTW between two sequences is the sum of distances of their corresponding elements. The classic DTW algorithm employs dynamic programming to identify corresponding elements so that this distance is minimal. The Longest Common Subsequence (LCSS) is another measure used to solve the alignment problem and to detect outliers in time series [VHGK03]. LCSS determines the longest common subsequence between two sequences. The Optimal Subsequence Bijection (OSB) [LWKTM07] is yet another measure which, in contrast to DTW, creates a one-to-one correspondence between two subsequences. Another difference to DTW is that OSB allows skipping of elements.

2.1.4 Time-Series Anonymization

As mentioned in the introduction to this chapter, we evaluate our measure (Section 6.3.2) with several use-cases, one of which is based on time-series anonymization. We therefore provide work related to this field in the following.

Research has produced numerous anonymization techniques. See [FWCY10] for an overview. *Differential Privacy* is an intuitive measure of the risk of one's privacy when having personal data in a database [Dwo06]. [AC11] is an example of a privacy-preserving system compliant with differential privacy. Other work has addressed time-series anonymization. [PLKY07] proposes several schemes for time-series anonymization in a streaming context. [RSMP11] studies the problem of smart-meter time-series anonymization by filtering out low-power frequency components.

Others have analyzed the effect of anonymization on subsequent use of the data. As an example, the framework developed in [BKJB13] allows to quantify the economic and environmental effects of anonymization on local energy markets. [LDR06] argues that the quality of anonymized data should be measured based on the workload the data would be used for. We have however not found any work which explicitly investigates this effect.

2.2 Computer Energy Estimation

The energy consumption of a computing system can be monitored directly. This is done by measuring the energy consumption using common digital meters [GFS+10], custom-designed devices [SMPH05] or integrated hardware power sensors [Int]. In the case of large and heterogeneous computing centers, installing digital meters or power sensors at every subsystem (server, PC, etc.) is costly.

A related domain of significant interest in recent research is computer power characterization at the system and subsystem level. Part of recently developed power characterization models are based on collecting microarchitectural events using hardware registers. These models consider both subsystem [Jan01, MB06, BGM+12] and system [BJ12] levels, as well as virtualized environments [DMR10].

A drawback of power characterization models which use hardware registers is that these models are tailored to specific hardware. This makes such models less portable and general. Thus, in the case of large heterogeneous deployments of computing systems, this lack of genericity would require a significant effort for power consumption modeling and estimation of the entire deployment. Another issue is that the number of hardware performance events tends to be large [Riv08]. Moreover, only a small part of them can be measured at the same time [Riv08], due to the limited number of hardware registers. [ASWW05] proposes a solution, namely time-multiplexing different sets of events on the hardware registers. While this approach allows for a greater number of performance events to be monitored, it increases the overhead and reduces the accuracy.

Using high-level statistical information provided by the operating system avoids the need for specific detailed (low-level) hardware knowledge when designing power estimation models. Recent work has proposed power consumption models based solely on high-level performance or usage metrics provided by the operating system to maximize energy efficiency using various optimizations. Thus, [FWB07] uses CPU utilization in order to estimate the power consumption of large numbers of servers, reaching a mean error of 1% when considering groups of several hundreds of servers. The power consumption model proposed in [HDVC$^+$05] uses the expected load on a server cluster in order to estimate its power consumption. $JouleMeter$ [KZL$^+$10] is a solution for virtual machine power metering which infers the power consumption from resource usage at runtime. [BNdM12] proposes a model of the power consumption of idle servers.

The advantages of high-level black-box models are the low overhead, simplicity and relatively good accuracy. However, these models estimate full-system power consumption and do not allow for a more fine-grained repartition of the power consumption, such as per process or per application power consumption. Moreover, the majority of the models has been developed and tested on computer systems with a big share of static energy, such as servers [FWB07] or clusters of virtual machines [KZL$^+$10]. Our approach (Chapter 5) allows the easy integration of such black-box models.

Several works have considered application or process-level power estimation of computer systems. [DRS09] has introduced $pTop$, a process-level energy profiling tool for the Linux platform, while [CLS12] has introduced $pTopW$ – an enhanced Windows-based version of the same tool. The $pTop$-tool estimates process energy consumption indirectly through the process's resource utilization. First, the total energy consumption of each resource (CPU, Memory, Network, Disk) during a sampling period is estimated using a system-level power model based on OS-level utilization information. Subsequently, for every resource, the energy consumption of a process is calculated by multiplying the share of work the resource has spent on the given application and the total energy consumption of the resource during that period. The work that a resource spends on a process depends upon its type. In case of the CPU, the work that the CPU spends on a process is defined as the time spent on the application. In case of the networking infrastructure, the work is the volume of data sent and received by the application. The authors in [NBRS12] use a similar model in order to estimate process energy consumption. They go further and develop models for the estimation of the energy consumption of methods and threads within a process. Similarly to $pTop$ and $pTopW$, one of the main goals of the tool is to offer detailed power information about energy hotspots at the application level.

2.3 Change Detection

The goal of change detection is identifying significant changes of the data or of its parameters. Research has produced numerous methods for different types of change to be detected. These methods usually fall into one of the following categories: sequential analysis, maximum-likelihood estimation, kernel-based techniques and Bayesian analysis techniques.

CUSUM is an established sequential analysis method for change detection of the parameters of a probability distribution [Pag54]. It calculates a cumulative sum for the segment currently considered and issues a change alert once this exceeds a given threshold. [BG] uses a sliding window which is partitioned into buckets. Each bucket can contain several data points; it does so by storing their number and an aggregate of their values. Each time a data point is added to the window, it is put into a new bucket. When a certain number of buckets is reached, the two oldest buckets are merged. If the difference of the average values of two neighboring buckets exceeds a dynamic threshold, a change is reported and the last bucket is dropped. This dynamic threshold is computed for each comparison of two buckets. It depends on the difference of the numbers of data points of the two buckets.

A further research area is detecting changes using models describing the data. [GS99] is based on maximum likelihood estimation. It examines a data window to which data points are added step by step. In each step, it determines if the window can be split into two significantly different segments. Each segment then is approximated by fitting a model to it, and the error between the model and the data is determined. The point which minimizes this error for both segments is reported as change point. The models used are derived from base classes such as algebraic polynomials, radial, wavelet or Fourier. [TY06] describes a two-stage algorithm which combines outlier detection and change detection. In a first stage, the algorithm learns an auto regressive (AR) model from a given time series. For each data point of the time series, a score is obtained by calculating the loss, be it the logarithmic one or the quadratic one. An outlier results in an isolated high score, while changes manifest themselves as series of high scores. Smoothing the scores removes the outliers. The smoothed values from the first AR model are then used to learn another AR model in the second stage of the algorithm. The scores of the second model describe the probability for data points being change points. [STR10] uses Gaussian Processes to model and predict the current run length – the length of a time segment between two consecutive changes.

[DDD05] uses one-class support vector machines for change detection. For each data point of the time series, the immediate past subset $x_{t,1}$ and the immediate future subset $x_{t,2}$ are mapped into a feature space. A kernel method is used; it ensures that the mapped input space is a subset of a hypersphere with radius one, centered at the origin of the feature space. Support vector classification then finds hyperplanes in the feature space which separate the training vectors $\Phi(x_{t,1})$ and $\Phi(x_{t,2})$ from the center of the hypersphere. To decide whether a change point is present, the authors introduce a dissimilarity measure in feature space:

$$D_H = \frac{\overset{\frown}{c_{t,1}c_{t,2}}}{\overset{\frown}{c_{t,1}p_{t,1}} + \overset{\frown}{c_{t,2}p_{t,2}}}, \tag{2.1}$$

where $c_{t,1}$ and $c_{t,2}$ are the centers of the hypersphere sections intersected by the hyperplanes, and $p_{t,1}$ and $p_{t,2}$ are two points where the hyperplanes intersect the hypersphere. The arc represents the arc distance between the two points. If the dissimilarity measure exceeds a given threshold, a change point is reported.

[AM07] uses a Bayesian approach. It divides a time series into partitions and assumes that for each partition there is an i.i.d. probability distribution of the data values. Thus, the change points are the boundaries between the partitions. For each new data point, the algorithm estimates the probability distribution since the last change point and then computes the probability that the new point belongs to this distribution. When this probability drops suddenly, a change is reported.

Some methods compare the probability distributions of (subsequent) sequences of data. [SWJR07] features a test which checks if two datasets are sampled from the same underlying distribution using a Gaussian kernel density estimator. [LYCS13] uses a density estimator to instead calculate the ratio of the distributions of two consecutive subsequences of data and to detect if they come from different distributions.

3 Datasets

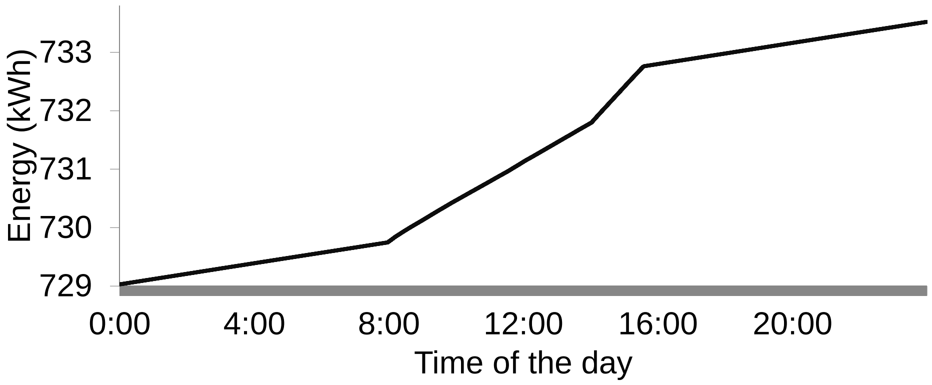
We have used multiple real-world datasets for evaluating our contributions. We present in the following their descriptions. The first four datasets contain energy consumption data of buildings or offices (Office, REDD, Smart*, Campus). The other three datasets (Server, Desktop, Laptop) contain energy consumption and usage data of computers.

3.1 Office Dataset

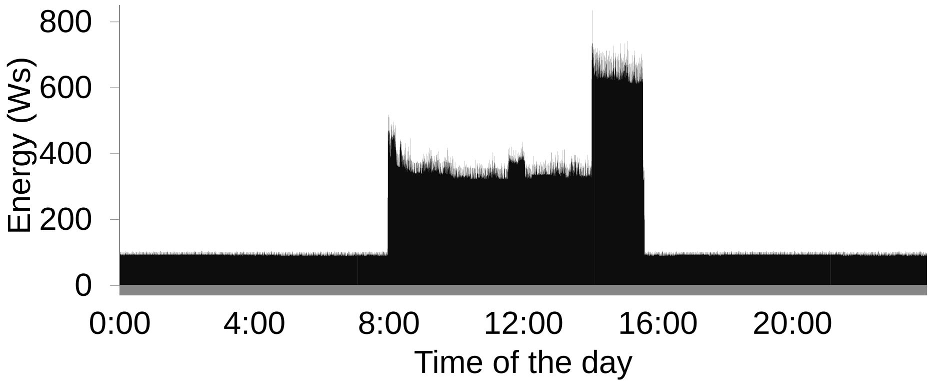
This dataset is from a smart meter installed in an office of two persons [VWSK10]. Measurements are done every second. Each measurement contains the amount of energy consumed up to the moment in time when the measurement was taken. This is, the values measured are monotonically increasing. By subtracting 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 3.1). We use the cumulative representation for the price-estimation scenario and the standard representation in the other two scenarios (Sections 4.2.2) for the evaluation of our compression method (Section 4.1).

3.2 The Reference Energy Disaggregation Dataset

The Reference Energy Disaggregation Dataset (REDD) was assembled by researchers in the field of energy disaggregation [KJ11] 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 4.2.3 for an example of the energy consumption in the REDD in standard representation.



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



(b) *Standard representation* derived from Figure 3.1(a) – each point refers to the energy consumed in the interval between the previous and the current point.

Figure 3.1: Two different data representations (one day, office dataset).

3.3 Smart* Home Data Set

This is the Smart* Home collection of datasets [BMI$^+$12], which we refer to as *Smart** throughout this dissertation. 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.

3.4 Campus Dataset

Besides the three rather fine-grained office, REDD and Smart* datasets, the campus dataset[dat10] 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.

3.5 Server Dataset

The **Server Dataset** is about a mail server executing SpamAssassin. Its workload is a daily pattern with a low usage during the night and a high usage in the morning and afternoon hours. Load peaks occur when the server checks bulks of e-mails sent to large mailing lists. Table 3.1 shows the hardware components of this system. P-states are power-performance states of the processor. We have used a digital multimeter Wattsup PRO [Wat15] (accuracy: 1.5%) to measure the energy consumption at every minute as a reference. Furthermore, our monitoring application has logged CPU usage, CPU frequency and hard disk drive usage with a sampling frequency of one second. Our measurements cover a period of three weeks.

3.6 Desktop Dataset

The **Desktop Dataset** contains three weeks of energy consumption, CPU usage and CPU frequency measured on an office computer (Table 3.2) with a sampling frequency of one second. Its workload is the result of typical secretarial tasks, e.g., MS Office, Internet Explorer and administrative applications. The

Table 3.1: Server Components

Component	Model	Power Consumption
CPU	2 x AMD Opteron 275	Maximum – 95.2 W P-State #1 – 90.3 W P-State #2 – 75.9 W Minimum P-State – 36.1 W Halt Mode – 16.6 W
Memory	Micron Technology 4x1 GB DDR400 PC3200	Minimum – 9.9 W Typical – 36.4 Maximum – 87.48 W
Hard Disk	2 x Seagate ST937401 2x74 GB at 10000 rpm	Maximum – 10.2 W Idle – 5.07 W Minimum – 4.69 W

workload rarely reaches the maximal computing capacity, and the computer is active only during office hours.

Table 3.2: Desktop Components

Component	Model	Power Consumption
CPU	Intel Pentium Dual Core E5300	Deeper Sleep – 4 W Extended Halt – 8 W Thermal Design Power – 65 W Maximum – 92.9 W
Memory	Crucial Memory 2x2 GB DDR2 SDRAM 800 MHz	Minimum – 3.65 W Typical – 5.1 W Maximum – 10.4 W
Hard Disk	Western Digital WD2500AAJS 250 GB 7200 rpm	Standby – 0.73 W Sleep – 0.73 W Idle – 4.92 W Read/Write – 5.36 W

3.7 Laptop Dataset

For the **Laptop Dataset** we have measured the same parameters as for the desktop dataset, over a period of two weeks. The laptop (Table 3.3) has been used for research purposes, i.e., the system load does not follow any regular pattern and shows idle periods as well as maximum load conditions.

Table 3.3: Laptop Components

Component	Model	Power Consumption
CPU	Intel i5-3320M	Idle – 2.9 W Minimum active – 7.5 W Thermal Design Power – 35 W Maximum active – 80.56 W
Memory	Micron Technology 2x4 GB DDR3L SDRAM 800 MHz	Minimum – 0.3 W Typical – 1.48 W Maximum – 1.68 W
Hard Disk	Hitachi HTS725050 500 GB at 7200rpm	Sleep – 0.1 W Standby – 0.2 W Active idle – 1.0 W Read/write – 1.8 W

4 Storage-Space Reduction

In this chapter we present our first contribution. We first revisit the motivation behind our adaptive time-series compression method. We then present and analyse its compression algorithm. Subsequently, we present the evaluation of our approach. We next describe our methods for estimating the compression ratio of our compression method, followed by their evaluation. We finally conclude the chapter with a summary of our contribution. Note that large parts of this chapter (figures, tables and algorithms included) are from our corresponding publication [EEKB15].

Time-series data is one of the most important types of data, and it is increasingly collected in many different domains [RLG+10]. In the domain of electrical energy, the advent of the Smart Grid leads to rapidly increasing volumes of time-series data [EPVM12]. It is therefore necessary to reduce storage requirements for time series. One solution is to decrease the frequency of capture, e.g, from one value per second to one value every 15 minutes. However, this impacts applications that need high-resolution data, such as disaggregation [KJ11]. Another solution is to employ classical lossless compression methods. However, these have not been designed for numerical time series and achieve in this case limited success [RRR+12, EEKB15]. As an example, the authors of the study in [RRR+12] have achieved the highest average compression ratio of factor 4 for datasets describing smart-meter readings from individual buildings using the bzip2 algorithm [Sal08].

For this reason, recent research has proposed several promising lossy compression methods [EEC+09, LM03]. These achieve good compression ratios by segmenting the time-series and approximating the segments with mathematical functions (piecewise regression). However, the performance of such approaches depends strongly on the nature of the data and it is unclear which method to use for a given dataset. Moreover, most approaches use one class of mathematical functions, e.g., constant functions. They therefore do not adapt well to highly variable time series. To address this issue, the authors in [PRA11] have presented the idea of employing not one, but several regression models for online time-series compression with a given quality guarantee. 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, [PRA11] 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.

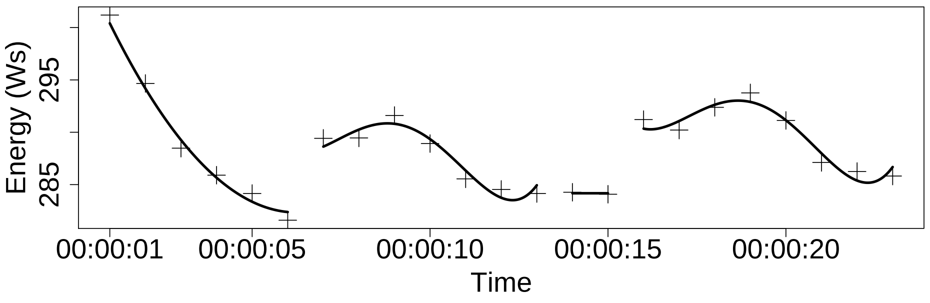


Figure 4.1: Example piecewise compression using regression functions.

We build on the idea of employing multiple models (Figure 4.1), 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 [DL06, LM03, Sei91] to approximate segments of time series using polynomial functions of different degrees. All of them provide guarantees under the uniform norm.

Our algorithm uses polynomials up to a certain degree p. Choosing this parameter depends on the dataset and impacts the performance of the compression. To find a good value of p for a given dataset, we present two methods which estimate the compression ratio depending on p. In a nutshell, our estimation methods use a statistical model of the average length of the segments resulting from piecewise regression. We then use them to predict the compression ratio for different values of p. We finally set p to the value for which our algorithm achieves the biggest compression ratio.

4.1 Algorithm

We now describe our compression algorithm (Section 4.1.1), discuss the selection of regression functions (Section 4.1.2), specify how to store compressed data (Section 4.1.3) and say how we compute compression ratios (Section 4.1.4).

4.1.1 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 than or equal to two. The PMR-Midrange algorithm [LM03] 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 [DL06] 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 [Sei91] 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 9). 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 10). We then pass to the polynomial of the next degree and repeat the procedure (Lines 5 to 9). 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 4.1.4 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 that 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 [LM03] 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 [DL06] 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 [Sei91] 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 4.2 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 [PRA11], 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. Our evaluation (Section 4.2) confirms this.

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 [LM03] 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. Our evaluation (Section 4.2) upholds this hypothesis.

4.1.2 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 [TZ83].

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 4.2.3). 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.

4.1.3 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 4.1.4), 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 [PRA11] 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 4.1). 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 4.1): 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 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.

Table 4.1: $COMPR\_SEG$ – 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

4.1.4 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 4.2.3). 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 4.1 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:

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

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 4.2.3, we rely on the storage scheme presented in Section 4.1.3. 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 and in line with related work, we assume a size of 64 bits for every data type in Table 4.1. 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.

4.1.5 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 4.2.2 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 4.2.3 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.

4.2 Evaluation of Storage-Space Reduction

In this section, we describe our experimental evaluation. At first, we present the datasets we use (Section 4.2.1). We then introduce three scenarios for the evaluation (Section 4.2.2). After that, we present and discuss the results (Section 4.2.3). We then discuss and evaluate one important aspect of our compression technique, the approximation of additional points (Section 4.2.4). Finally, in Section 4.4 we evaluate our methods for estimating compression performance.

4.2.1 Experimental Setup

We have used the following datasets in our evaluation:

- The **Office Dataset** (Section 3.1)
- The **Reference Energy Disaggregation Dataset (REDD)** (Section 3.2)
- The **Smart Home Dataset (Smart)** (Section 3.3)
- The **Campus Dataset** (Section 3.4)

4.2.2 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.

Price-Estimation Scenario

Giving consumers access to their consumption data measured by smart electricity meters can result in increased energy efficiency [MSW10]. Our scenario assumes that energy prices are dynamic, i.e., electricity prices which are different at different points in time [EPVM12]. 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 [Gmb06]. 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 [Gmb06]. We choose the maximum tolerable deviation to be less than 5%, i.e., ± 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 ± 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.

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 4.2, which shows the typical power consumption of several standard household appliances [oE13]. As we can observe, 25 Ws is significantly smaller than the power consumption of most standard home appliances.

Table 4.2: 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

Energy Forecast Scenario

Forecasting is a key technique in the smart grid (see Section 2.1.2). Here, 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 [DBF$^+$10].

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 [DBF$^+$10]:

- 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 [HK06], is given by:

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

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

4.2.3 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 4.1.4 in this section. In realistic deployments of the compression technique, where exact storage requirements are known, the values for compression ratios might differ.

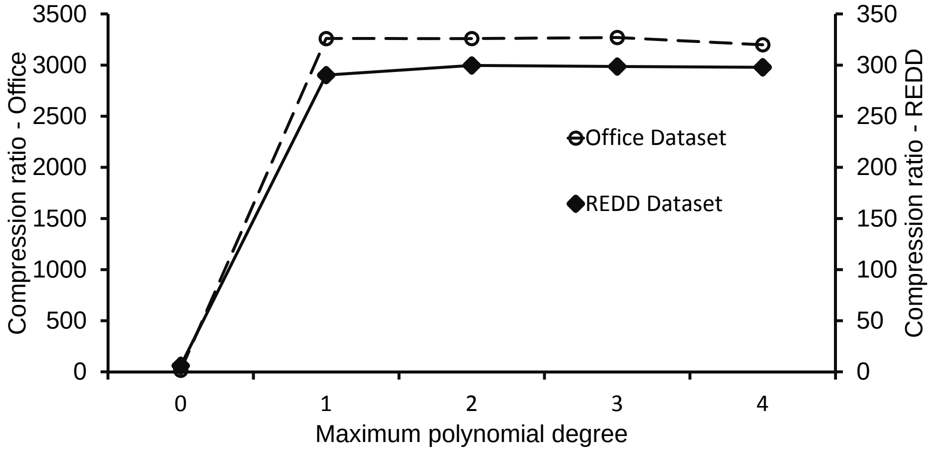
Price-Estimation Scenario

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 4.2(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.

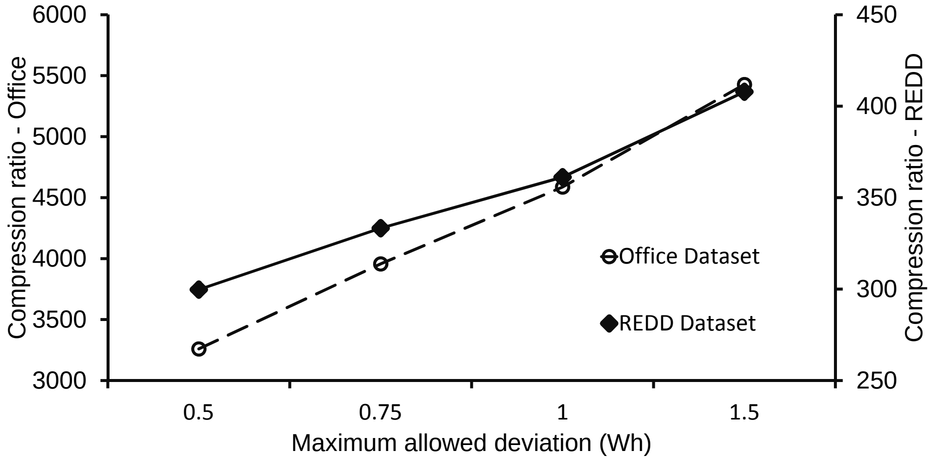
It is interesting to note that the compression ratio can also slightly decrease when including polynomials of degrees higher than three. This is because the algorithm uses a greedy approach. Consider the following example result of our compression algorithm: There are two consecutive intervals, the first 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 4.2(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



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



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

Figure 4.2: Compression results, price-estimation scenario.

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 4.3 presents the average runtimes of the compression algorithm on the REDD.

	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

Table 4.3: Average runtime (seconds) per day (REDD; price estimation).

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.

Visualization Scenario

We again firstly present the results describing the compression ratio depending on various maximum degrees (p) of the polynomial functions. Figure 4.3(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.

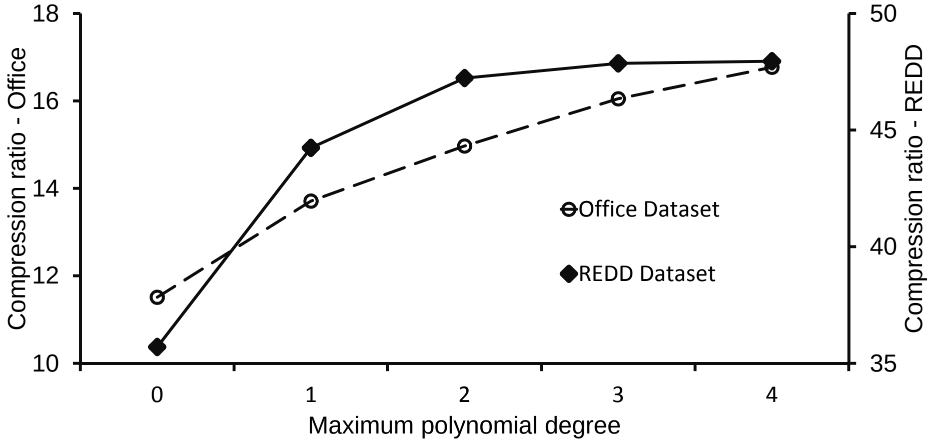
Figure 4.4 illustrates the compressed and decompressed versions of a time series of the REDD. In this case, the maximum deviation allowed is of 25 Ws, corresponding to a compression ratio of factor 108 (see Figure 4.3(b)). We can observe that with our highest deviation tolerable, apart from a smoothing 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 4.3(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).

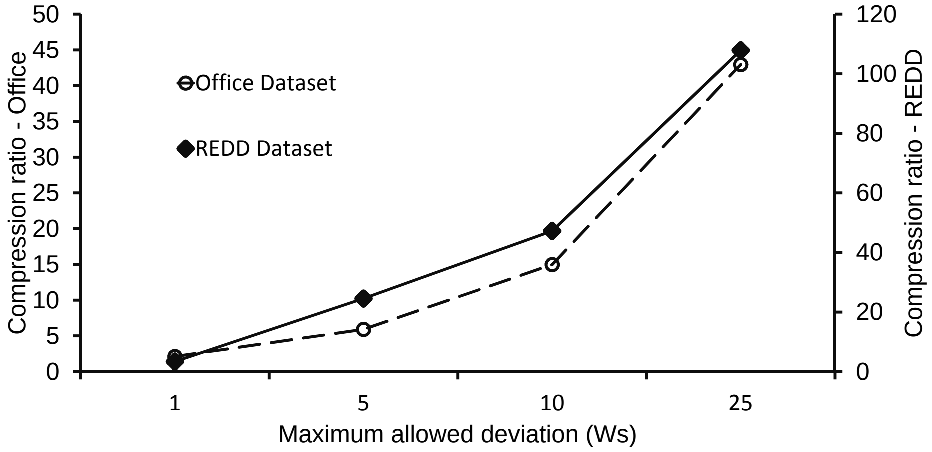
In general, it is notable in this scenario that compression ratios are lower with the office dataset than with the REDD. However, the situation has been the other way round in the price-estimation scenario. This can be explained by the different nature of the two datasets and the different choice of 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.

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



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



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

Figure 4.3: Compression results, visualization scenario.

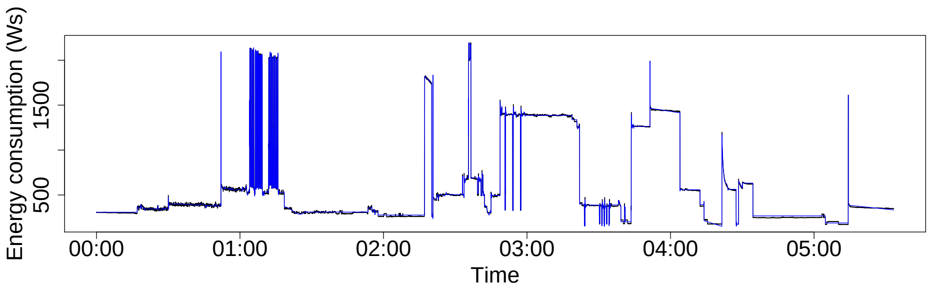


Figure 4.4: Visual comparison of two curves corresponding to original (black) and compressed (blue) data of the REDD.

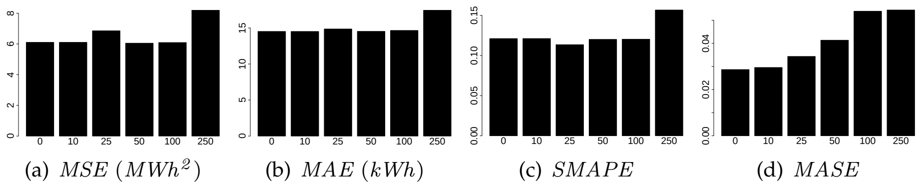


Figure 4.5: Energy forecast results on compressed data using four error metrics (the horizontal axis depicts the maximum allowed deviation in Wh).

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 4.2.2) in comparison to the original uncompressed data and have averaged them at the end.

Figure 4.5 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.

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

Table 4.4: Compression ratios on the campus dataset (forecast).

Table 4.4 shows the values of the compression ratios obtained for the different values of the maximum deviation allowed, corresponding to the error metrics in Figure 4.5. 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.

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 [LM03] ('constants' in Table 4.5) and straight-line functions [EEC$^+$09] ('lines' in Table 4.5). We do not compare our compression technique to techniques based on Fourier [FRM94] or wavelet [CF99] decomposition. As mentioned in Section 2.1.1, these techniques do not provide any guarantee for the compressed version of the data.

Table 4.5 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

(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

Table 4.5: Compression ratios compared to related work.

is used), 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 4.4(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 4.4(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 [PRA11] (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.

4.2.4 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 [Mit97], 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.

4.3 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 [LM03], the second one uses disconnected straight-line functions – the slide filter introduced in [EEC$^+$09]. 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 4.4.

4.3.1 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.

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 [BJ76], which tends to produce less autocorrelated time series. This also happens when applying this transformation to the time series in our datasets (Section 4.4). Table 4.6 shows the autocorrelation values for the original and first-difference time-series of two energy consumption datasets (see Section 4.2.1). We use the following formula [BD02] to calculate the autocorrelation of a time-series X_t with mean μ at lag h:

$$\rho_X(h) = \mathbb{E}[(X_{t+h} - \mu)(X_t - \mu)] \tag{4.1}$$

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 4.6. 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.

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.

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

		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

Table 4.6: Autocorrelation values for original and first-difference time-series of energy consumption data.

and min are the maximum and minimum values of the segment [LM03]. 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: $\frac{max-min}{2} \leq \epsilon$ (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:

$$\sum_{i=1}^{n} D_i = (X_1 - X_0) + (X_2 - X_1) + \cdots + (X_n - X_{n-1}) = X_n - X_0 \qquad (4.2)$$

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 $2 \cdot \epsilon$:

$$max(S(D,n)) - min(S(D,n)) \leq 2 \cdot \epsilon \qquad (4.3)$$

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:

$$\begin{aligned}\Pr(Z = n) = \Pr(&max(S(D,n)) - min(S(D,n)) \leq 2 \cdot \epsilon \wedge \\ &max(S(D,n+1)) - min(S(D,n+1)) > 2 \cdot \epsilon)\end{aligned} \qquad (4.4)$$

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:

$$\mathbb{E}[Z] = \sum_{i \geq 1} \Pr(Z > i) = \sum_{i \geq 1} \Pr(max(S(D,i)) - min(S(D,i)) \leq 2 \cdot \epsilon) \tag{4.5}$$

To approximate $\mathbb{E}[Z]$ we use the following lemma, which is based on results from [Fel51]:

Lemma 4.3.1. *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 4.5 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 $[0, 2 \cdot \epsilon]$ 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 $\epsilon$ be the predefined error bound
2: Let $\delta = 0.001$
3: Let $i = 1$
4: Let $add = \Pr(Z > i)$
5: Let $estimation = add$
6: while $add > \delta$ do
7:     $add = \Pr(Z > i)$
8:     $estimation = estimation + add$
9:     $i = i + 1$
10: end while
```

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 4.6 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.

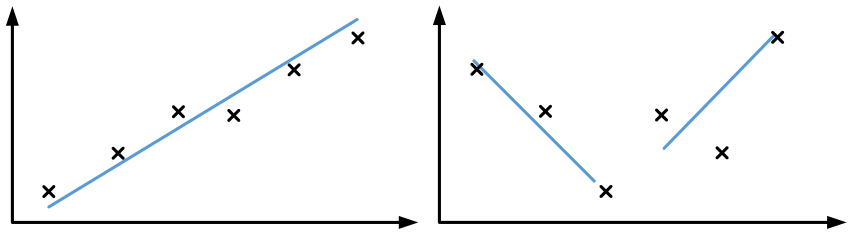

(a) Time series 1 (b) Time series 2

Figure 4.6: 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:

$$\sum_{1}^{i} D_i = X_n - X_0 = X_n \tag{4.6}$$

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 $(1, X_1 + \epsilon)$, $\ldots$, $(i, X_i + \epsilon)$. The variable $LB_{low,i}$ is the largest slope of all lines passing through the point $(0, X_0)$ and one of the points $(1, X_1 - \epsilon), \ldots, (i, X_i - \epsilon)$. 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 4.7 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 $i \geq 2$ using the following recursive formulas:

$$LB_{up,i} = min(LB_{up,i-1}, \frac{X_i + \epsilon}{i}) \tag{4.7}$$

$$LB_{low,i} = max(LB_{low,i-1}, \frac{X_i - \epsilon}{i}) \tag{4.8}$$

with

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

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:

$$LB_{up,i} \geq LB_{low,i} \tag{4.9}$$

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 4.3.2. *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.*

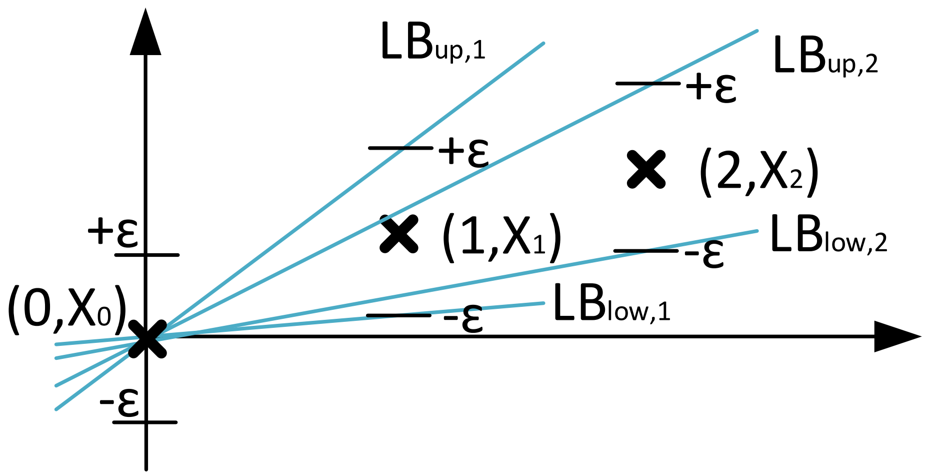


Figure 4.7: Estimation using straight-line functions – lower bound.

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:

$$\Pr(Z_{low} = n) = \Pr(LB_{up,n} \geq LB_{low,n} \ \wedge LB_{up,n+1} < LB_{low,n+1}) \tag{4.10}$$

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:

$$\mathbb{E}[Z_{low}] = \sum_{i\geq 1} \Pr(Z_{low} > i) = \sum_{i\geq 1} \Pr(LB_{up,n} \geq LB_{low,n}) \tag{4.11}$$

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:

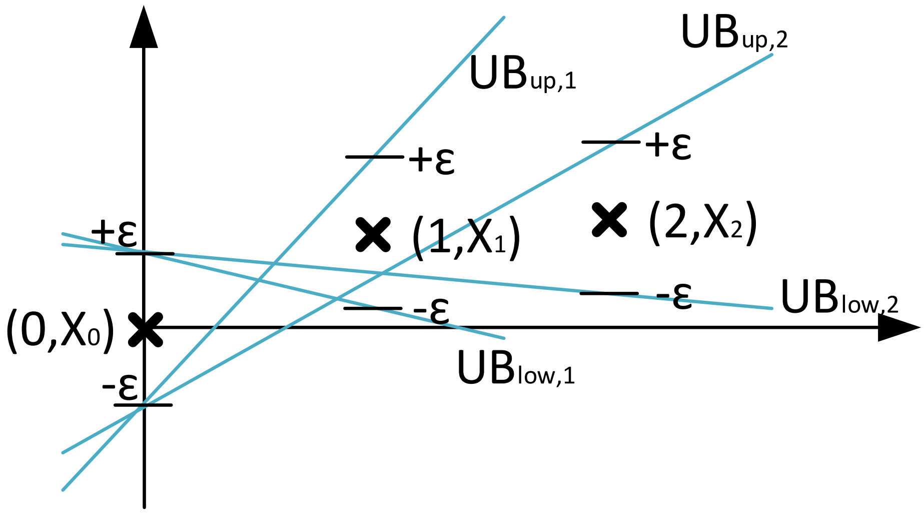


Figure 4.8: Estimation using straight-line functions – upper bound.

$(0, X_0 - \epsilon)$ and $(0, X_0 + \epsilon)$. 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 $(0, X_0 - \epsilon)$ and one of the points $(1, X_1 + \epsilon)$, ..., $(i, X_i + \epsilon)$. Similarly, $UB_{low,i}$ is the largest slope of all straight lines passing through the point $(0, X_0 + \epsilon)$ and one of the points $(1, X_1 - \epsilon)$,..., $(i, X_i - \epsilon)$. Figure 4.8 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 $i \geq 2$:

$$UB_{up,i} = min(UB_{up,i-1}, \frac{X_i + 2 \cdot \epsilon}{i}) \tag{4.12}$$

$$UB_{low,i} = max(UB_{low,-1}, \frac{X_i - 2 \cdot \epsilon}{i}) \tag{4.13}$$

with

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

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:

$$UB_{up,i} \geq UB_{low,i} \tag{4.14}$$

If Condition 4.14 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 4.3.3. *Let a sequence of data points* $[(t_0, X_0), (t_1, X_1), \dots, (t_m, X_m)]$ *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:

$$\Pr(Z_{up} = n) = \Pr(UB_{up,n} \geq UB_{low,n} \ \wedge UB_{up,n+1} < UB_{low,n+1}) \tag{4.15}$$

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:

$$\mathbb{E}[Z_{up}] = \sum_{i \geq 1} \Pr(Z_{up} > i) = \sum_{i \geq 1} \Pr(UB_{up,n} \geq UB_{low,n}) \tag{4.16}$$

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.

4.3.2 Generation-Based Estimation

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

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.

Algorithm

To efficiently determine the length of the time series we generate, we use the method originally described in [DL06]. 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).

4.4 Evaluation of Estimation Methods

In the following we present an evaluation of our estimation methods described in Section 4.3. 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

Algorithm 3 Generation-based estimation

```
1: Let $\epsilon$ 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 $current\_error < \epsilon$ do
11:        $current\_seg = add\_next\_points(D, j)$
12:        $current\_error = approximate(current\_segment)$
13:        $j = 2 \cdot j$
14:    end while
15:    $l = perform\_binary\_search(current\_seg, j)$
16:    $time\_series\_complete.add(ts)$
17: end for
18: $estimate = get\_average\_length(time\_series\_complete, N)$
```

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:

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

4.4.1 Evaluation of Runtime Performance

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 4.7 shows the runtimes of both methods for the case of constant functions for one random time series of the REDD dataset (see Section 4.2.1). 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.

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 4.7: Runtime Comparison – Constant Functions (seconds)

Table 4.8 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.

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

Table 4.8: Runtime Comparison – Straight-line Functions (seconds)

4.4.2 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 4.3, 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 4.9 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.

	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 4.9: Constant functions, real and estimated values, REDD house 2.

Table 4.10 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%.

	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 4.10: Constant functions, real and estimated values, Smart* home C.

Concerning all the time series in each of the datasets, Table 4.11 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.

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

Table 4.11: Constant functions, average APE.

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 4.12 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.

	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

Table 4.12: Straight-line functions, real, lower and upper bound values, REDD house 3.

Table 4.13 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 maximum 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.

	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

Table 4.13: Straight-line functions, real, lower and upper bound values, Smart* home B.

We present in Table 4.14 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.

		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

Table 4.14: Straight-line functions, average accuracy (%) of lower and upper bounds.

4.4.3 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 4.15 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.

	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 4.15: Generation-based Estimation – Constant Functions – Real and estimated values – REDD house 4

Table 4.16 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%.

	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

Table 4.16: Generation-based Estimation – Constant Functions – Real and estimated values – Smart* home B

Regarding all time series in each of the datasets, Table 4.17 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.

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

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

Table 4.17: Generation-based Estimation – Constant Functions – average APE

model-based estimation. However, as pointed out in Section 4.4.1, 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 4.18 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.

	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 4.18: Straight-Line Functions – Real and estimated values – REDD house 1

Table 4.19 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.

We present in Table 4.20 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.

	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

Table 4.19: Straight-Line Functions – Real and estimated values – Smart* home B

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).

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

Table 4.20: Generation-based Estimation – Straight-Line Functions – average APE.

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 4.21 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.

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.

		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

Table 4.21: Generation-based Estimation – Polynomial Functions – Average APE

4.4.4 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 4.22 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.

		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

Table 4.22: Decision accuracy for all datasets tested

4.5 Summary

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 chapter, 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. 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.

5 Computer Energy-Consumption Estimation

In this chapter we describe our second contribution. We begin by supplementing the motivation for estimating computer energy consumption, which we presented in the introduction. We then explain our contribution in detail: we describe three application scenarios for energy estimation and two respective effort classes. We then present our framework for computer energy estimation. Subsequently, we show how we evaluated our framework using three application scenarios and real-world datasets. We finally conclude the chapter with a summary of our contribution. In this section, we reused and combined large parts of our publications [EBB14a] and [EBB14b], including figures and tables.

Capturing time series data is useful for many analytical purposes, e.g., monitoring of residential power quality [IKG12] or visualization [NSQ+12]. Capturing computer energy consumption is particularly interesting: In recent years, the share of energy consumed by computers has significantly risen. Thus, Vereecken et al. [VVHC+10] has estimated the share of energy consumed by computers to be 7.15% of the total electricity consumption, and it is estimated to grow further (approx. 14.6% by 2020). Thus, the energy consumption of IT systems is an important cost driver for any enterprise, and quantifying this type of consumption reliably is a cornerstone for many current business models. For example, the energy consumption has a significant impact on the total costs of ownership of a data center. It must be considered for total absorption accounting. Furthermore, in the context of the smart grid, that data gives way to new business models, e.g., by scheduling data centers according to an oversupply of renewable energies.

Although useful, capturing this data involves in many cases sensors a high effort, e.g., smart electricity meters, must be acquired and installed for measuring and communicating the data [GFS+10, SMPH05]. This is expensive in large scales. In the case of computer energy consumption, it is however possible to estimate the data instead of measuring it directly. Recent research has provided methods to *estimate* the energy consumption of computer hardware, e.g., based on nameplate information [PN11], sophisticated hardware models [NBRS12], or by profiling system and component power usage [RRK08, KZL$^+$10]. However, all of these approaches have different characteristics in terms of setup effort, estimation effort, estimation accuracy and hardware requirements. It is difficult to decide when to use which approach and how to determine meaningful estimation parameters, as well as the accuracy requirements of a given application.

To address these challenges, we devise FRESCO, a FRamework for the Energy eStimation of COmputers, which considers many different accuracy and effort measures. This is difficult, because today's computer systems come with a wide range of different usage parameters and technical specifications, and we have to consider use cases that differ very much in the accuracy required and the effort acceptable for the operator. FRESCO consists of a configurable set of estimators, and a workflow to set up and run an instance of an estimator. In particular, FRESCO is able to (a) suggest a set of appropriate estimators for computer energy consumption according to the effort the operator is willing to invest and to the requirements of a certain application, and (b) to execute an instance of the selected estimator with settings that are appropriate for the application. Depending on the kind of estimator, FRESCO can estimate the energy consumption of a computer from various parameters. Such parameters include (1) hardware characteristics, e.g., the energy consumption of a hard disk as specified by its vendor, (2) usage information like CPU load and network activity, and (3) calibration data, e.g., an energy consumption profile that has been recorded by an energy meter for a specific hardware.

FRESCO explicitly models the trade-off between the accuracy of the estimation and the effort of obtaining technical specifications, building energy profiles or measuring CPU usage information. For example, FRESCO can estimate the energy consumption of a PC with a high precision in hourly intervals, but also with a lower precision in intervals of a few seconds. Furthermore, FRESCO can provide upper and lower bounds for the estimation, and it considers heterogeneous hardware components and heterogeneous loads.

5.1 Application Scenarios

In this section, we describe three different use cases that cover the spectrum of energy-aware applications for FRESCO.

5.1.1 Energy-Aware Management of Data Centers

Increasing the performance per watt is a key performance optimization for data centers [RASRK07, Lau05]. For this purpose, it is important to obtain the energy consumption of a complex IT system as early as the design time of the data center or the allocation time of the various computing workloads. Recent approaches, e.g., in the area of energy-aware cloud data centers [MCLRB09] or energy management for warehouse-sized computing centers [FWB07], distinguish (1) the (static) energy consumption at idle state, and (2) the (dynamic) energy consumption depending on the workload of the target system. This is important to design the power distribution infrastructure, to decide about computing hardware acquisitions or to find out if a scheduled workload exceeds the cooling capacity.

Thus, two different accuracy requirements exist: It must be possible (a) to provide estimates for the typical case that are sufficiently accurate to make educated decisions for hardware acquisitions, and (b) to provide upper bounds for the energy consumption in extreme cases. Both requirements must be fulfilled at design time or at allocation time, i.e., before the operator can measure the workload or the energy consumption. Furthermore, an estimator must consider that some in-depth hardware specifications might be unavailable at design time.

5.1.2 Demand-Response

Demand Response (DR) contains measures that influence energy-consumption patterns. For example, DR might be used to shift energy-intensive computing tasks to times of an energy surplus [PD11]. DR can be divided into (a) incentive-based DR and (b) time-based rates DR [PD11]. Incentive-based DR measures shift the energy consumption by providing, say, tariffs that reward to shift energy consumption into off-peak hours. In contrast, time-based rates DR makes use of static schedules.

Since a data center is a large, adjustable energy sink, it is particularly well suited to perform demand response measures [BLFdM13]. To realize DR in a data center, an estimator must deliver continuous estimates of the energy consumption of the various IT components at run time. In particular, the estimates must be adequate to identify system states that produce energy-consumption peaks. Furthermore, the personnel costs and computational effort of the estimation must not exceed potential savings from DR, otherwise the use of DR is not justified. Finally, the estimator must cope with technical parameters on different levels of detail. For example, a coarse estimate could measure the average CPU load only, while a fine-grained approach might also consider the voltage and frequency of the CPU and the states of other hardware components.

5.1.3 Computer Energy Accounting and Billing

Energy accounting and billing of the IT infrastructure becomes more and more important. For example, in the context of total absorption accounting, an enterprise might wish to assign each benefactor (a good or a service) the energy costs required for its production [JGC+11].

Typically, computer Energy Accounting requires estimates of the consumption with a frequency of 15 minutes to one hour. Furthermore, the estimator must provide stochastic accuracy guarantees (e.g., an accuracy of $\pm$ 10%), that allows to assign the energy consumption of an IT system to a department or a product line. As for the previous scenarios, the estimator has to be applicable to a large variety of computer systems, with an effort that is adaptable.

5.2 Classes of Effort

We have identified two classes of effort, which our framework must take into account.

The **Setup Effort** is necessary to set the estimator up and running. This includes collecting technical specifications of the energy consumption of certain hardware components, e.g., the energy consumption of the CPU in activity states like idle or sleeping. Furthermore, it contains the effort of installing a monitoring application to measure run-time parameters of the hardware usage, e.g., disk activity. Finally, the setup effort includes the calibration of an energy consumption profile for a given hardware, e.g., measuring the energy consumption while executing a benchmark application.

The **Run-Time Effort** includes the network overhead and the computational overhead of the estimation process, and the overhead of a monitoring application collecting hardware parameters like CPU frequency or rotation speed of the hard disks, if required by the estimator. The more parameters the estimator samples, the higher are the data volume transferred, the computational overhead and the complexity of the estimation.

We expect that the two effort classes will be traded for each other. For example, the same accuracy requirement can be met either (a) by measuring an energy consumption profile at setup time or (b) by using detailed specifications and usage parameters at run time.

5.3 Framework

In this section we describe the workflow and the estimators of our FRamework for the Energy eStimation of COmputers.

5.3.1 The FRESCO Workflow

With *"Target System"* we refer to the computer system whose energy consumption FRESCO must estimate. "The *Operator*" is responsible for installing and maintaining the estimator on the target system. FRESCO consists of three subsequent stages *"Setup"*, *"Configuration"* and *"Estimation"*, as shown in Figure 5.1, which we explain in the following.

Setup At the first stage, the operator quantifies the trade-off between effort and estimation accuracy for the target system. In particular, the operator specifies the categories of information obtainable from the target system. This includes:

- The nameplate information available, e.g., if the energy consumption of the network card can be obtained.
- The parameters measurable on the target system, e.g., CPU frequency, CPU voltage or hard disk activity.
- If it is possible to measure a consumption profile for the target system, and with which accuracy.

The type of estimates and their error guarantees influence the choice of the estimators FRESCO suggests.

At the end of the setup stage, FRESCO either indicates the operator that, given his input, estimation is impossible or lets the operator choose one or a combination of estimators. The former case happens, e.g., when the operator requires estimates with stochastic guarantees while calibration is not possible. In the latter case, FRESCO provides information on the estimation accuracy possible and the effort required for each estimator.

Configuration At this stage, FRESCO helps the operator to configure the estimators selected, according to:

- The usage parameters that must be measured to meet the accuracy specified in the setup stage.
- The frequency at which the parameters must be measured.
- The energy consumption profile, if necessary.

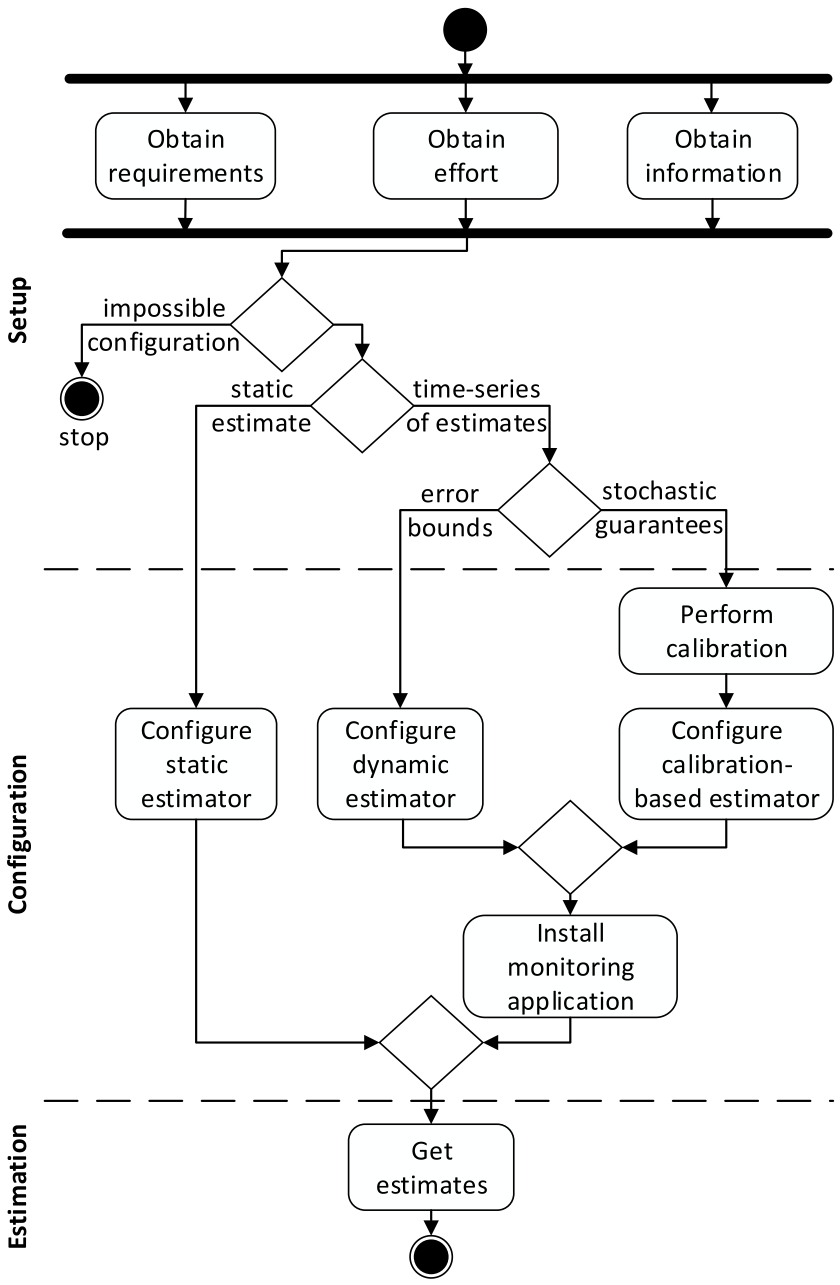


Figure 5.1: FRESCO Workflow

If the operator has chosen a calibration-based estimator, FRESCO provides a set of benchmarks and guides the operator through measuring the energy consumption. The result of this stage is the combination of configured estimators.

Estimation Finally, FRESCO runs instances of the chosen estimators with the configuration parameters just fixed on the target system and estimates its energy consumption.

5.3.2 The FRESCO Estimators

FRESCO can use static, dynamic or calibration-based estimators or a combination of them. The *Static estimator* makes use of static information on the computational load and hardware specifications. The *Dynamic estimator* predicts the energy consumption from hardware specifications and run-time parameters measured. The *Calibration-based estimator* uses an energy consumption profile that has been calibrated at the configuration stage. In the following, we describe each estimator, and we discuss its effort and accuracy.

Static Estimator

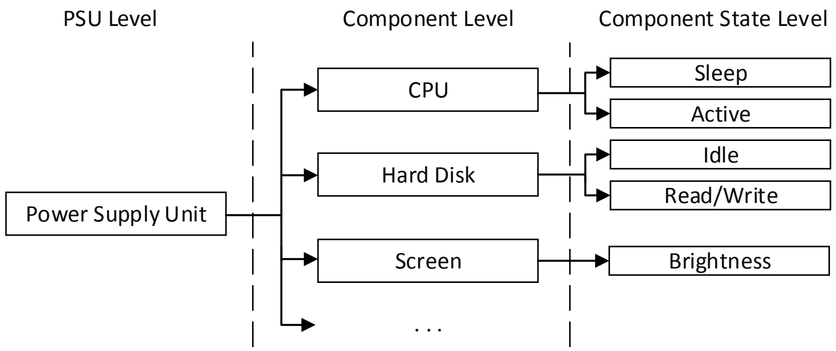


Figure 5.2: Classes of Information Used by the Static Estimator

Our static estimator is inspired by [PN11]. The approach in [PN11] is tailored to servers running a single application, similarly to the TPC (Transaction Processing Performance Council) benchmark suite [tpc], at peak load. Thus, it aggregates the peak power consumption of all hardware components. Since we are interested in estimates for a wide range of target systems operating at different loads, we have extended this approach. In particular, we have modeled a wide range of computer architectures and hardware components, e.g., laptop screens or motherboards of standard PCs. Furthermore, we consider three levels of detail to model the computer architecture, namely the PSU level, the component level and the component state level, as shown in Figure 5.2. We have obtained these levels of detail by abstracting from the parameter sets used in [PN11]. For the component-state level we consider all power-consumption states besides peak consumption.

The *PSU level* considers only the specification of the Power Supply Unit (PSU) powering the target system. In this case, the estimator calculates the total energy consumption E by using the maximum power PSU_{max} the PSU is able to supply and the run time ΔT of the target system:

$$E = \Delta T \cdot PSU_{max} \tag{5.1}$$

We use PSU_{max}, the only value available at this level of detail. If a higher accuracy is required, if more information is available, and if more effort at setup time is acceptable, FRESCO considers information at the *component level*. In this case, the consumption E is the sum of the power consumptions P_i of each component $i \in C$, $C = \{CPU, RAM, Hard\ Disk, ...\}$, multiplied with the run time ΔT:

$$E = \Delta T \cdot \sum_{i \in C} P_i \tag{5.2}$$

Note that we represent the CPU of a multicore system as a set of components. One component constitutes core-independent consumption, e.g., due to caches shared by all cores. The other components represent individual cores.

The *component state level* includes information on (1) the power consumption of the different states of the components of the target system and (2) the time the components typically spent in certain states. In this case, FRESCO obtains the total consumption E by summing up the energy consumption P_{ij} of each component $i \in C$ in a particular state $j \in S_i$ (the set of states of component i), multiplied with the time ΔT_{ij} each component typically is in this state:

$$E = \sum_{i \in C} \sum_{j \in S_i} \Delta T_{ij} \cdot P_{ij} \tag{5.3}$$

In the case of a virtualized environment, the static estimator incorporates virtual machines at the component state level, i.e., the operator maps components to virtual machines.

The accuracy of a static estimator depends on the detail level used and on confidence intervals on the input parameters. In the following, we will briefly discuss two extreme cases:

Component	Model	Power Consumption
CPU	Intel i5-3320M	Idle – 2.9 W Minimum active – 7.5 W Thermal Design Power – 35 W Maximum active – 80.56 W
Memory	Micron Technology 2x4 GB DDR3L SDRAM 800 MHz	Minimum – 0.3 W Typical – 1.48 W Maximum – 1.68 W
Hard Disk	Hitachi HTS725050 500 GB at 7200rpm	Sleep – 0.1 W Standby – 0.2 W Active idle – 1.0 W Read/write – 1.8 W

Table 5.1: Laptop Components

Minimal information: By using Equation 5.1, the operator obtains a very coarse upper bound of the energy consumption of the target system. This is because the PSU intake, as described on its nameplate, is usually overestimated for safety reasons [FWB07]. However, if the manufacturer has provided tolerance bounds for the PSU intake in typical settings, the operator might be able to narrow down this upper bound.

Full information: Each hardware manufacturer provides detailed data sheets containing the minimal, typical and maximal energy consumption of any hardware component. If the operator specifies parameters on the component state level (cf. Equation 5.3), this information can be used to obtain hard upper and lower bounds for the energy consumption.

Example 1. *Consider a laptop as described in Table 5.1. The lower bound on the consumption is the sum of the minima of each component, i.e., 2.9 W + 0.3 W + 0.1 W = 3.3 W. Likewise, the upper bound is the sum of the maximum values: 80.56 W + 5.5 W + 1.68 W = 87.74 W.*

Summary: *A static estimator might be sufficient for any application that does not need time series of estimates. It requires a small effort at setup time for obtaining the hardware specifications, and no effort at run time. The accuracy of this estimator depends on the detail level of its input values and the availability of tolerance bounds. In particular, the static estimator can provide bounds on the energy consumption.*

Dynamic Estimator

Our dynamic estimator models the energy consumption similarly to the static estimator, but installs a monitoring application on the target system to periodically measure detailed load information in real-time, e.g., CPU load or sleep times of the hard disk. Thus, our dynamic estimator generates time series of energy consumption data. Our dynamic estimator uses a monitoring application to record at run time in which state $j \in S_i$ the component $i \in C$ operates at time t. The energy consumption E_t at time t is the sum of the consumptions P_{it} of the components i:

$$E_t = \sum_{i \in C} P_{it} \tag{5.4}$$

The consumption E for a time interval $[t_p; t_q]$ is the sum of the consumptions at each point in time, multiplied with the period of time Δt between taking two consecutive samples:

$$E = \sum_{i=t_p}^{t_q} E_i \cdot \Delta t \tag{5.5}$$

The energy consumption P of the components can be modeled in different ways. For example, consider the consumption P_t^{CPU} of the CPU. Suppose that the monitoring application measures the state information "CPU load" l_t^{CPU}, and the operator knows the minimum and maximum power P_{min}^{CPU} and P_{max}^{CPU} the CPU can consume. In this case, P_t^{CPU} is:

$$P_t^{CPU} = P_{min}^{CPU} + l_t^{CPU} \cdot (P_{max}^{CPU} - P_{min}^{CPU}) \tag{5.6}$$

It is also possible to integrate specific models for multi-core systems [BdM12] and to model virtual machines as components of the target system. While the accuracy of the static estimator depends on the knowledge about the typical load of the target system, our dynamic estimator samples such parameters. Thus, the accuracy of our dynamic estimator depends on the sampling frequency of the monitoring application. The reason is as follows: If a state changes between taking two consecutive samples, the estimator does not know to which extent the states were active. However, FRESCO provides upper and lower bounds on the energy consumption by assuming that a state change has taken place immediately before or after taking a sample. The upper (lower) bound is the maximum (minimum) of the power consumptions of the two consecutive samples.

Example 2. *Suppose that the hard disk of a laptop (Table 3.3) has been observed in standby at time t_3, and as idle at t_4. Thus, the lower bound on the consumption in interval $[t_3 : t_4]$ is $0.2W \cdot \Delta t$, and the upper bound is $1.0W \cdot \Delta t$.*

Formally, consider a component with the sequence of states $S = (s_1, s_2, \cdots, s_n)$, ordered by the power P_i, the component consumes in state i. Let $s_t = (s_{t_1}, s_{t_2}, \ldots)$ be the time series of the states of the component sampled at times $t_1, t_2, \ldots$. The upper bound $E^u_{\Delta t}$ on the energy consumption during time interval Δt between consecutive samples t_j and t_{j+1} is:

$$E^u_{\Delta t} = \begin{cases} P_1 \cdot \Delta t \text{ if} & s_{t_{j+1}} = s_1 \wedge s_{t_j} = s_1 \\ \ldots & \\ P_i \cdot \Delta t \text{ if} & s_{t_{j+1}} = s_i \wedge s_{t_j} \leq s_i \vee \\ & s_{t_j} = s_i \wedge s_{t_{j+1}} \leq s_i \\ \ldots & \\ P_n \cdot \Delta t \text{ if} & s_{t_{j+1}} = s_n \wedge s_{t_j} \leq s_n \vee \\ & s_{t_j} = s_n \wedge s_{t_{j+1}} \leq s_n \end{cases}$$

We calculate the lower bound $E^l_{\Delta t}$ likewise.

Summary: *Our dynamic estimator is suitable for applications that require time series of the energy consumption of the target system at run time. Since this estimator also needs technical specifications, it requires a similar effort at setup time as a static estimator. The effort at run time depends on the number of parameters that the estimator samples, and on the sampling frequency. The accuracy of our dynamic estimator depends on the technical specifications and the sampling frequency. The estimator can compute bounds on the energy consumption.*

Calibration-Based Estimator

This estimator borrows from the Mantis approach [ERK06], which estimates the power consumption of a system by correlating AC power measurements from a calibration phase with performance counters of the CPU. Our calibration-based estimator executes a detailed benchmark at setup time, which gradually stresses each system component in isolation. At the same time, a digital power meter records the actual energy consumption, and our monitoring application measures load information such as CPU frequency, hard disk usage, etc. FRESCO then builds a regression model.

More specifically, let $M^{CPU}(l, f), M^{Disk}(l), M^{RAM}(l)$ be the regression models obtained through calibration for the CPU, Hard Disk and RAM, and let l be the load of the component and f the frequency of the CPU. Given the load information l^{CPU}_t, l^{Disk}_t, l^{RAM}_t and f_t at time t, the energy E consumed in the time interval $[t_1, t_n]$ is:

$$E = \sum_{i=t_1}^{t_n} \left(M^{CPU}(l^{CPU}_i, f_i) + M^{RAM}(l^{RAM}_i) + M^{Disk}(l^{Disk}_i) \right) \tag{5.7}$$

The calibration-based estimator can derive stochastic accuracy guarantees from the regression model used, by considering the maximal and minimal energy consumption that has been recorded for each distinct benchmark load. For example, think of a regression model which uses only CPU load l and CPU frequency f to estimate the total energy consumption. Let (f_c, l_c) be the pair of values for the current load and frequency of the CPU and $E_{l_c,f_c} = \{e_1, e_2, ..., e_m\}$ be the set of values for the real energy consumption that the calibration phase had measured for the pair (f_c, l_c). The upper bound $E^u_{\Delta t}$ for the energy consumption during the time Δt when the CPU operates at frequency f_c and at load l_c is

$$E^u_{\Delta t} = \max E_{l_c,f_c} \cdot \Delta t \tag{5.8}$$

while the lower bound $E^l_{\Delta t}$ is:

$$E^l_{\Delta t} = \min E_{l_c,f_c} \cdot \Delta t \tag{5.9}$$

We calculate upper and lower bounds for other pairs of values of frequency and load the same way.

Example 3. *Assume that the benchmark has resulted in a CPU load of 50% and in a frequency of 2.4 GHz for some time interval, and the energy consumption measured has been {97.5 W, 99 W, 100 W, 102 W, 102.5 W, 103 W}. Thus, for this load and frequency FRESCO would stochastically guarantee a maximal (minimal) energy consumption of* $103W \cdot \Delta t$ *(*$97.5W \cdot \Delta t$*). Since the dynamic estimator and the calibration-based estimator use the same monitoring application, it is possible to obtain upper and lower bounds for the energy consumption from our dynamic estimator in tandem.*

Summary: *The calibration-based estimator is well-suited for applications that require a calibrated zero point and stochastic guarantees on the estimation quality, such as billing. Due to the extensive calibration, this estimator comes with a very high effort at setup time. The calibration can be done automatically. Its effort is necessary only once during setup and depends on the number of components calibrated and their type. As an example, the calibration of the CPU of the laptop computer (Table 3.3) took around 25 minutes. This particular CPU can operate at 16 different frequencies. At run time, the effort depends on the number of parameters that must be sampled and on the sampling frequency.*

Summing up everything, FRESCO can model a wide range of computer architectures and environments, including multi-core and virtualized systems. It can generate static ex-ante estimates solely on hardware specifications as well as time series of estimates at run time with a configurable estimation frequency and different accuracy guarantees. FRESCO considers different kinds of effort at setup time and run time. In the next section, we evaluate FRESCO.

5.4 Evaluation

Our framework operates as intended if its estimates are appropriate for a wide range of applications. That is, FRESCO must let the operator decide on a trade-off between accuracy and effort, according to the requirements of the application. Thus, we evaluate FRESCO by means of three use cases, and we measure the accuracy obtainable with a certain effort.

5.4.1 Measures

To evaluate how well FRESCO can estimate the real data measured by our digital multimeter, we have computed two metrics. The **Mean Absolute Percentage Error (MAPE)** measures the average of the percentual deviation between the actual values and the estimates:

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - y_i'|}{y_i}$$

where y_i are the actual values and y_i' are the estimates. n is the number of records in the dataset. MAPE of zero means that the estimated values perfectly match the ones measured. Correspondingly, the **Maximum Absolute Percentage Error (MaxAPE)** measures the maximum percentual deviation between the actual values and the estimates:

$$MaxAPE = \max_{i=1}^{n}\left(\frac{|y_i - y_i'|}{y_i}\right)$$

5.4.2 Evaluation Setup

We have evaluated three different datasets:

- The **Server Dataset** (Section 3.5)
- The **Desktop Dataset** (Section 3.6)
- The **Laptop Dataset** (Section 3.7)

5.4.3 Use Cases

We have evaluated FRESCO with the three use cases described in Section 5.1.

Energy-Aware Data Center Management

Our first use case (cf. Subsection 5.1.1) requires ex-ante estimates of the energy consumption depending on a predefined workload. The estimates must be sufficiently accurate for informed management decisions, e.g., it must be possible to find out if one target system requires significantly more energy for a certain workload than another one. Furthermore, it must be possible to find out if a certain workload may exceed the cooling capacity in the worst case. Thus, FRESCO proposes the static estimator model. To evaluate this scenario, we let FRESCO estimate upper and lower bounds on the energy consumption, and the average energy consumption for a typical workload.

Minimal and Maximal Consumption We let FRESCO exemplarily estimate upper and lower bounds on the consumption of a server. With our use case, the operator specifies manufacturer information on the component level for CPU, RAM and disk, as specified in Table 3.1. Our target system consumes the most energy when the CPU operates with the highest frequency at the highest voltage allowed in the specifications (95.2 W), and when the hard disk is in read/write mode at the highest rate of I/Os per second (10.2 W). The RAM consumes at most 87.48 W. Our server has two CPUs and two disks. Thus, the upper bound for the energy consumption is (2·95.2 + 2·10.2 + 87.48) W = 298.28 W. Correspondingly, the lower bound for the power consumption is 52.48 W. Our measurements confirm that these bounds apply. The bounds can be narrowed if the operator is able to specify a limit for the time each component is in a specific state (cf. Equation 5.3).

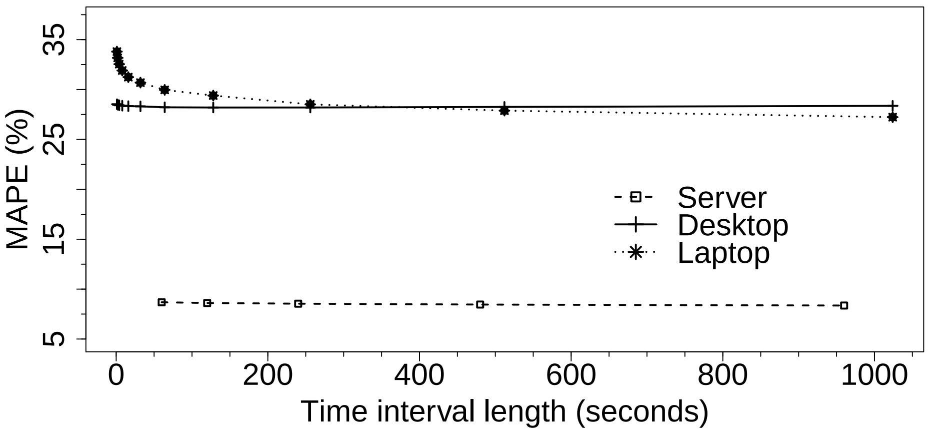


Figure 5.3: MAPE, Static Estimator

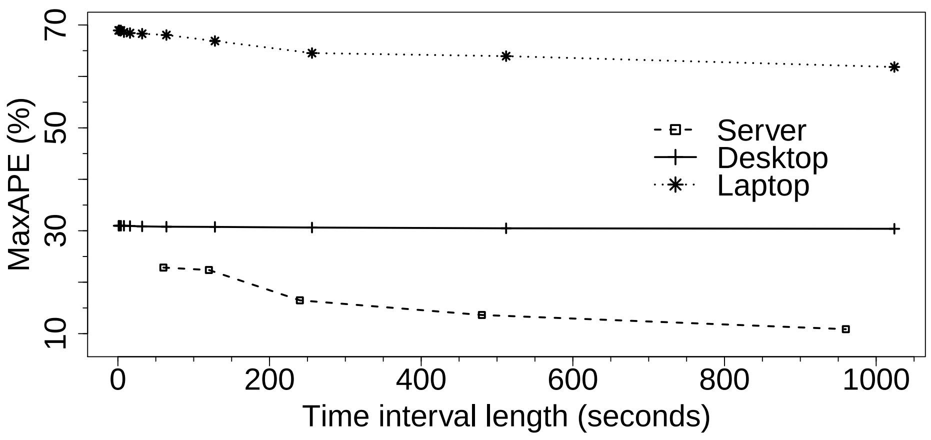


Figure 5.4: MaxAPE, Static Estimator

Average Consumption Now we assume that the operator wants FRESCO to estimate the energy consumption for an average system load of 50%, in order to assess the typical cooling requirement. To evaluate the accuracy of the estimates, we aggregate the energy consumption, which we have measured with a frequency of up to one second, to time intervals from one second to 16 minutes. Furthermore, we let FRESCO use the static estimator to provide estimates for the same time intervals. Figure 5.3 shows the MAPE on the y-axis and the length of the time interval on the x-axis, for each of our three datasets. A value of 30% at the interval length of 1 minute for the laptop dataset means that, on average, the energy consumption estimates summed up for intervals of 1 minute deviate by 30% from the corresponding consumption measured. Figure 5.4 shows the MaxAPE for all datasets. The figures indicate that for longer time intervals, FRESCO can provide more accurate estimates. In particular, for the server dataset, the maximum error goes from around 23% for an aggregation level of one second to around 11% for a higher aggregation level of 16 minutes, corresponding to a two-fold decrease in value. Smaller decreases (69–62% and 31–30%) occur for the other two datasets. This is because longer interval lengths mitigate the effect of short-term deviations in the workload. Estimates depend on the operator approximation of the average load. Thus, the results for the laptop dataset show that the operator has under- or overestimated the average load significantly. Evidently, the accuracy of the estimation can be improved if the operator provides more accurate information on the workload of the target system.

Summary: *FRESCO has provided upper bounds for the energy consumption. Furthermore, it is able to provide reasonable estimates for the average workload by requiring only little data from the operator. Thus, we conclude that FRESCO is able to deal with* *the requirements of this use case.*

Demand Response

This use case (cf. Subsection 5.1.2) requires time series of estimates to identify periods of time with high energy consumptions (peaks), together with upper and lower bounds. As the operator is willing to invest only a small effort, FRESCO suggests our dynamic estimator model.

To evaluate this scenario, we let FRESCO estimate (cf. Equation 5.6) the consumption based on the CPU load and on information on the maximal and minimal energy consumptions of our three target systems with a frequency of one second. We use these estimates to identify points in time when the energy consumption is above a given threshold. In particular, we evaluate two dynamic thresholds that consider the difference between the largest and smallest values of a time series T:

$$\theta_1 = 0.8 \cdot \big(\max_{i=1}^{|T|}(T_i) - \min_{i=1}^{|T|}(T_i)\big) \tag{5.10}$$

$$\theta_2 = 0.95 \cdot \big(\max_{i=1}^{|T|}(T_i) - \min_{i=1}^{|T|}(T_i)\big) \tag{5.11}$$

We compute time series of peak consumption from our measured values as well as for the time series FRESCO has estimated, by filtering out all values that are smaller than θ. If our estimates are accurate, FRESCO can identify periods with high energy consumption and can thus enable operators to perform Demand Response.

Figure 5.5 illustrates the cumulative distribution function (CDF) of the real energy consumption during specific intervals for the desktop dataset. The first set of intervals is when FRESCO estimated the consumption to be greater than θ_1 (continuous line). The second set is when FRESCO estimated the consumption to be greater than θ_2 (dashed line). We observe that, if the estimator predicts a value greater than θ_1, then the real energy consumption is greater or close to θ_1. Thus, in around 88% of all cases, a value predicted to be greater than θ_1, is also greater than θ_1. In 80% of all cases where a value greater than θ_2 was estimated, the real energy consumption was greater than 90% of θ_2.

Figure 5.6 illustrates the cumulative distribution function (CDF) of the predicted energy consumption during specific intervals for the desktop dataset. The first set of intervals is when the real consumption was greater than θ_1 (continuous line). The second set is when the real consumption was greater than θ_2 (dashed line). Thus, the estimator predicted a value of at least 75% of θ_1 for

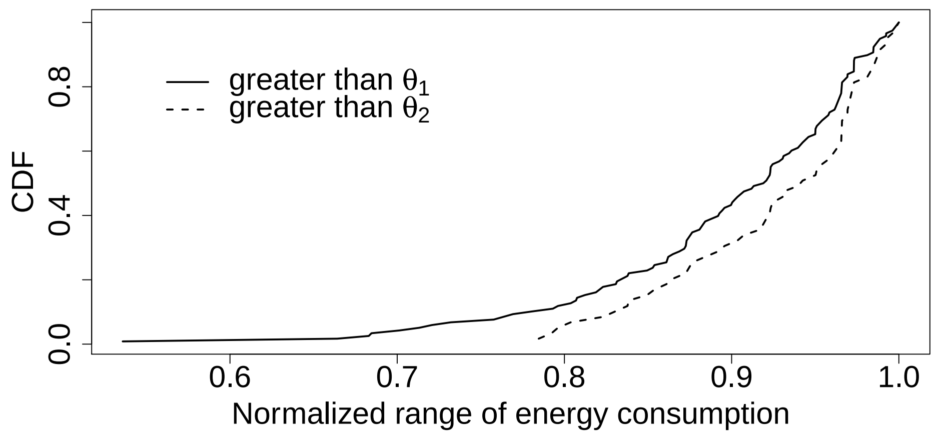


Figure 5.5: CDF, Desktop Dataset

values which were greater than θ_1 in 90% of the cases. The estimator predicted a value of at least 80% of θ_2 for all values which were greater than θ_2 in all cases.

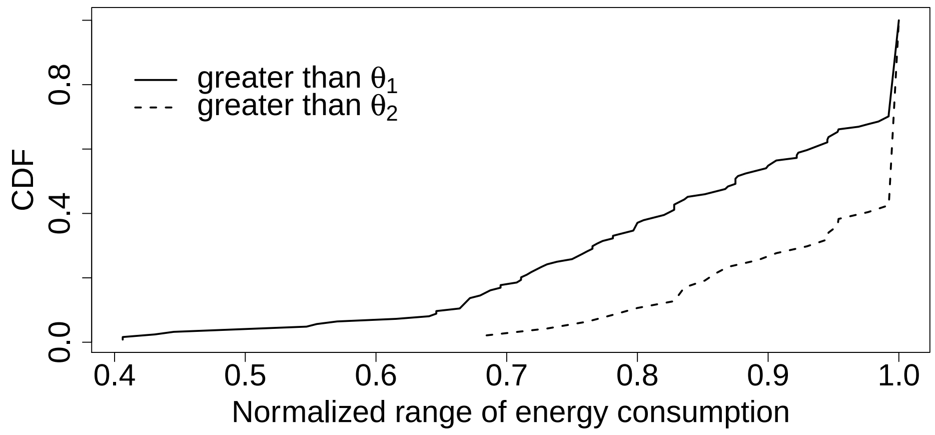


Figure 5.6: Distribution of Predicted Energy Consumption Values – Desktop Dataset

For the laptop dataset (Figure 5.7), if our dynamic estimator predicts that the energy consumption during an interval is greater than θ_1, then the actual consumption is greater than 75% of θ_1 in 92% of the cases. Furthermore, a value greater than θ_2 of the estimated energy consumption is greater than 75% of θ_2 of the real consumption in around 92% of the cases. On the other hand (Figure 5.8), our estimator predicted a value of at least 75% of θ_1 for intervals with a consumption greater than θ_1 in 95% of the cases. Our estimator predicted values greater than 80% of θ_2 for all intervals with a consumption greater than θ_2.

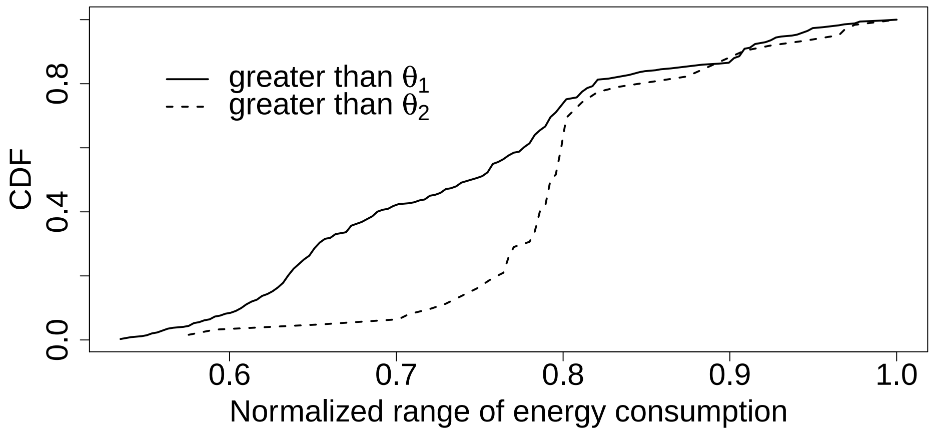


Figure 5.7: Distribution of Real Energy Consumption Values – Laptop Dataset

Concerning the server dataset, for which we use minute-by-minute measurements and estimations, the accuracy of our dynamic estimator is better. In 90% of the cases where an estimate is greater than θ_1, the real value also is greater than θ_1. Additionally, the estimator correctly predicted all values greater than θ_2. On the other hand, our estimator predicted a value of at least 75% of θ_1 for intervals with a consumption greater than θ_1 in 90% of the cases. Our estimator predicted values greater than 80% of θ_2 for all intervals with an actual consumption greater than θ_2.

Summary: *Our dynamic estimator can identify periods of time with peak energy consumptions with a reasonable accuracy with a low estimation effort. That is, it uses only static information on the minimal and maximal consumption of the target system, and it samples only the CPU load. Moreover, the estimator is flexible, i.e., it can sample more usage parameters in order to improve its accuracy. We conclude that FRESCO fulfills the requirements of this use case.*

Energy Accounting

Our third use case (cf. Subsection 5.1.3) requires estimates with stochastic guarantees. Thus, FRESCO suggests a calibration-based estimator, which provides estimates based on calibrated energy profiles.

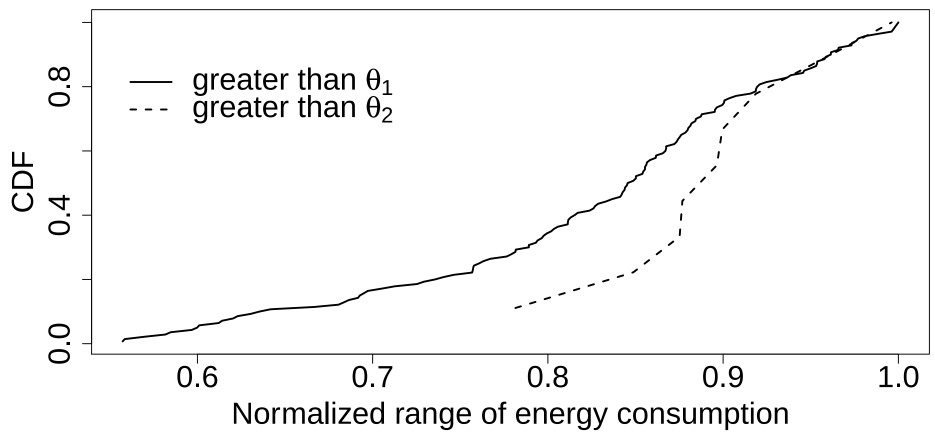


Figure 5.8: Distribution of Predicted Energy Consumption Values – Laptop Dataset

We let FRESCO calibrate energy profiles for each of our three target systems at setup time. At run time, our monitoring application samples the CPU load with different sampling frequencies. In order to compare the estimates with the real values, we have calculated MAPE and MaxAPE for all three datasets. Figure 5.9 shows the MAPE on the y-axis and the length of the time interval on the x-axis. The figure indicates that the estimation accuracy is better for longer time intervals. For all three datasets, the mean error decreases by around a fourth (14–37% decrease) when the estimator aggregates estimates for intervals of 16 minutes instead of one second. Similarly, the MaxAPE decreases significantly for all datasets with longer estimation intervals, as shown in Figure 5.10. In particular, for the server dataset, the maximum error is around 6.5 times smaller, decreasing from about 35% to 5.3%. For the other two datasets, the decrease in maximum error is significant as well (2.4 times for the laptop dataset and 22 times for the desktop dataset, respectively).

Summary: *With estimation intervals that make sense in energy accounting, FRESCO is able to provide estimates of a high accuracy. While the effort at run time is similar to the one of the dynamic estimator, the effort at setup time is very high. However, many energy accounting scenarios make use of numerous target systems with identical hardware, e.g., for typical office tasks. In such scenarios, the calibration effort at setup time takes place only once. Thus, we conclude that FRESCO can perform well in an energy accounting scenario.*

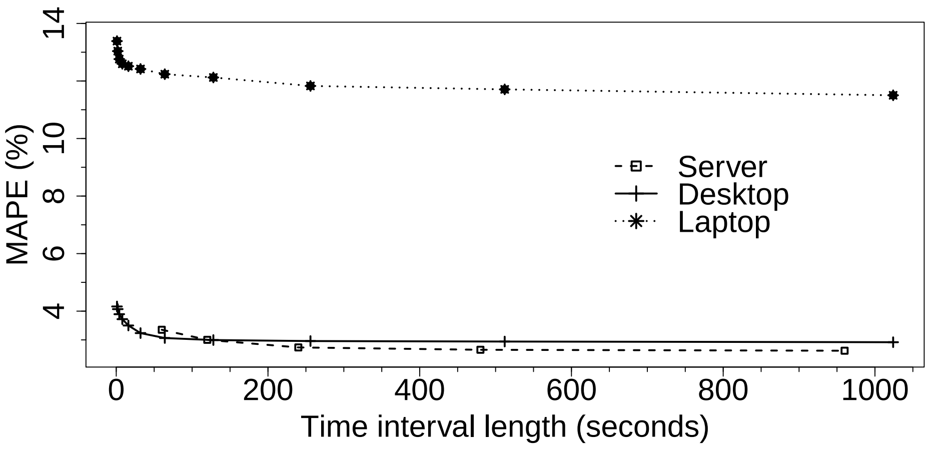


Figure 5.9: MAPE, Calibration-Based Estimator

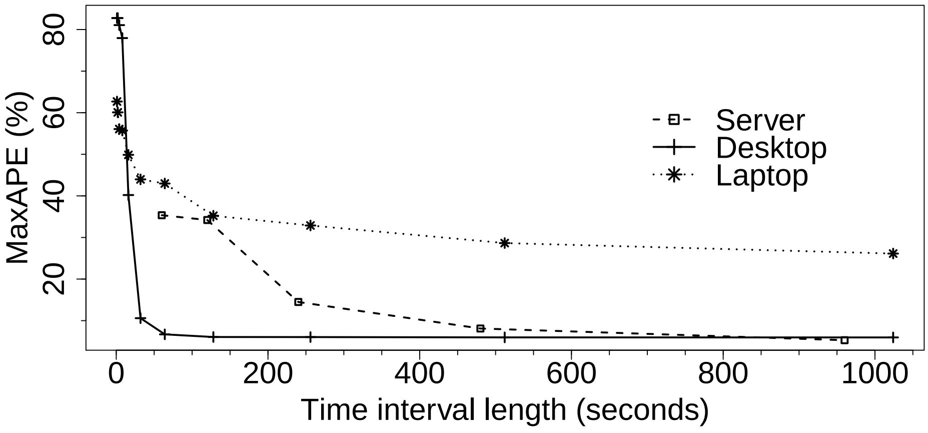


Figure 5.10: MaxAPE, Calibration-Based Estimator

5.5 Summary

The effort of collecting fine-granular time series is high as often data must be measured and communicated using sensor nodes, e.g., smart electricity meters. This is expensive in large scales. In certain cases, it is possible to obtain time series of estimates instead of direct measurements. This is in the case of computer energy consumption particularly interesting. One reason is that, as the share of computer energy consumption increases, it becomes increasingly important to quantify it in a solid manner. Many enterprise applications, accounting procedures and business models require such data to allow informed management decisions, e.g., for energy-aware management of IT resources, IT-energy accounting or demand response. However, most existing estimators are tailored to specific use cases, hardware architectures and usage profiles. Due to their different characteristics in terms of effort and accuracy, the choice of estimation method for a given application is far from obvious. In this chapter, we have proposed FRESCO, a general and flexible FRamework for the Energy eStimation of COmputers. Depending on the effort the operator is willing to invest and on the requirements of the application, FRESCO can propose and run appropriate estimators with good parameter settings. It gives quality guarantees on the estimates. FRESCO considers heterogeneous hardware components and loads, as well as the frequency of the estimation. Experimental results based on three representative real-world datasets show that our framework is useful in many business use-cases.

6 Lossy Transformations and Subsequent Change Detection

In this chapter we describe our third contribution, which builds on the first two. We start with the motivation for quantifying the impact of lossy transformations on subsequent data analysis. We then describe three application scenarios using different types of lossy transformations and having different requirements on the quantification. We then present MILTON, our Measure for quantifying the Impact of Lossy Transformation methods for time series on subsequent change detectiON. We further explain how we calculate and parameterize MILTON. Subsequently, we show how we evaluated our measure using three application scenarios and real-world datasets. We finally conclude the chapter with a summary and outlook of our contribution. In this chapter, we reused our corresponding publication [EBEB15], including figures, tables and algorithms.

Approaches, such as lossy compression [EEKB15], estimation [EBB14a] or perturbation/anonymization [PLKY07] lossily transform the time series: A lossy transformation can reduce the data volume, generate an optimized data model or remove personal information from a dataset. Such transformations however modify useful characteristics of the data, such as changes in the case of time series. Change detection on time series data is an important building block of many real-world applications [Pag54, GS99]. It converts a time series of measurements into one of events. The impact of such transformations depends on the application scenario, and quantifying it is far from trivial.

Existing similarity measures for time series, applied to the original series and the compression result, cannot meaningfully quantify the impact of a lossy transformation on the result of a change-detection approach [BC94, RK05, VHGK03, LWKTM07]. Such a quantification however is needed to identify and parametrize a good compression algorithm or anonymization approach, given a certain dataset and quality requirements on the change-detection result. Think of energy-consumption data from a smart meter, which serves as our running example. Change detection on such data allows to detect interesting events (turning on/off of a device, abnormal device activity). Such events are needed for demand side management, peak shifting, peak shaping, etc. – all basic techniques to integrate renewable energy sources into the Smart Grid. However, data transformation, e.g., lossy compression or anonymization, can modify the data considerably. This can significantly impact the subsequent detection of those events.

Example 4. *An energy provider uses a lossy compression technique for time series from a smart meter, to reduce the data volume, before running a change-detection algorithm. Due to the compression loss, (a) some changes might be detected at different points in time, or (b) their significance might be altered, compared to the original time series. Next, (c) changes might go undetected at all, or (d) the compression might result in new changes. Using his domain knowledge, the provider can assess the importance of these impacts. Based on his assessment, he wants to select a concrete compression technique, together with a good parameter set.*

The quantification of this impact is difficult due to several open challenges: First, as shown in the example, the impact is manifold. One therefore needs to carve out possible effects of a lossy transformation on changes. Second, the definition of a measure for this impact is not obvious. It is necessary to thoroughly investigate application scenarios where one is working on the transformed data, in order to come up with respective requirements. Third, the measure envisioned should be customizable to the concrete application scenario. Think of the energy provider once again. For him, it will be more detrimental if compression eliminates changes from the data, as opposed to the insertion of new ones. Fourth, identifying the specific effect of a transformation on a change (e.g., shift in time vs. disappearance) is an application-dependent procedure, which must take all changes into account. This is because ascribing an effect to a certain change may cascade and influence the ascription of effects to other changes. Having defined a measure does not make finding a good algorithm computing it obsolete.

To address the challenges above, we design MILTON, *a practical and flexible Measure which quantifies the Impact of various Lossy Transformation methods for time series on subsequent change detectiON.* This measure is applicable to any use case where one wants to know how much a certain transformation approach for time series reduces the result quality of a change-detection technique, as compared to change detection on the original data. This lets an operator decide how much he can compress the data without affecting change detection considerably, or if the anonymization technique he intends to deploy does indeed conceal certain changes, as he had planned. To ensure flexibility, we do not impose restrictions on the change detection or the transformation approach used, and we allow to flexibly weight effects on changes.

At first sight, an alternative to MILTON would have been to derive a model of the loss of data quality due to a transformation. It is however very demanding to build such a model that is generally applicable. The main reason is that it is difficult to impossible to integrate each of the many existing lossy transformation techniques and change-detection approaches into one model.

6.1 Use Cases Involving Change Detection

In this section, we describe three scenarios which motivate our measure and derive the requirements on it. We have consciously decided to describe these scenarios in much detail, in order to reveal the subtle differences between them, which then give way to the requirements.

6.1.1 Data Compression in the Smart Grid

Description

The first use case involves using lossy compression techniques [EEKB15, HJA13, LM03, EEC+09] that produce a piecewise approximation of the original data within an error threshold ϵ. These techniques not only modify the original data, but also the changes present in it. The effects of such approximations on the changes have not been investigated yet. An energy provider intending to use the compressed data for analytics needs to take these effects into account. For instance, by detecting changes in data streams and integrating them in the learning model, he can improve forecasting [RLB13] or enhance stream mining [TO09]. We refer to this scenario as the “compression scenario”.

Problem Domain

The result of lossy compression methods depends on the models they use (e.g., constants, straight lines, polynomials) and how they use them. To evaluate their impact on changes in the data, one thus needs to consider different classes of models. Another important parameter here is the error threshold ϵ. Compression results, and consequently their impact on changes, strongly depend on this parameter. One therefore needs to evaluate the impact of compression methods for different values of ϵ.

Setting

The energy provider employs a forecasting application that uses the compressed time series to predict the energy consumption. Detecting a change in the time series triggers an update of the underlying model of the forecasting algorithm, to improve predictions. In such a case, it makes sense to penalize changes which disappear ("missed") more than those which emerge ("false positives") as a result of the transformation. This is because a missed change prevents the forecasting algorithm from updating its model when necessary. This may impact its accuracy significantly. A false-positive change in turn will trigger an unnecessary update of the model, which may cause additional effort, but should not affect forecasting accuracy considerably. Regarding shifts of changes in time, the provider deems them important for forecasting, as they will delay or vice-versa advance the update of the underlying model. On the other hand, modifications of the importance of changes are not crucial in this case, so he chooses to ignore them altogether. This makes sense here, because, once a change is detected, the model is updated regardless of that importance.

6.1.2 Data-Center Energy Management

Description

Estimates of computer-energy consumption are useful in many use cases (see Section 5.1). For instance, a data-center manager can use such data in the design phase of the data center or to keep track of the energy consumption of the IT infrastructure when operating the center [EBB14a]. He can additionally use the data for other use cases which employ change-detection methods including consumption-event characterization or detection of abnormal consumption. As with compression, estimates are an approximation of the real consumption data. A data-center manager thus needs to quantify the effect of estimation methods on changes and to choose an estimation method appropriate for subsequent change detection. We refer to this scenario as the "estimation scenario".

Problem Domain

Estimation methods function differently in order to obtain approximations. We are aware of several classes: A first class performs a sophisticated calibration process [FWB07], while another one relies on specific models of components [NBRS12]. They thus obtain estimates of different accuracy. In an evaluation, it would be interesting to study the impact of estimation methods from different classes on the changes. Another parameter here is the time granularity of the estimates. There is a trade-off between this granularity and accuracy [EBB14a]. We therefore need to evaluate estimation methods with different values for that parameter.

Setting

Here, the data-center manager will use estimates to balance energy demand and supply with the following application: A significant change in consumption will trigger an alarm, so that the energy supply adjusts to the new level. This means that shifts and modifications of the importance of changes are critical to the subsequent application. Regarding missed and false-positive changes, the manager uses a similar logic as in the previous scenario. A missed change is critical here because it prevents from balancing energy consumption and supply. A false positive however only implies an unnecessary readjustment. Even though this indicates additional costs, it is not critical to the subsequent application.

6.1.3 Data Privacy in the Smart Grid

Description

Smart meters can accurately and frequently measure and communicate the energy consumption of households. While these measurements are useful for analytical purposes, they unintentionally allow to infer personal information, such as the daily routine of residents [BBBK13, MMSF^{+}10]. The pseudonymization of the data is not sufficient. This is because an easy re-identification of consumers using simple statistical measures is possible [BBBK13]. Adding noise (e.g., white noise) to the data does not enhance privacy either, as one can easily filter it out [PLKY07]. One way to prevent filtering out noise is to first transform the data to another basis (e.g., apply a Wavelet transform), then to add noise to the data in that basis, and finally to re-transform the data to the original basis [PLKY07]. Although they do not guarantee anonymization, such methods give way to some extent of anonymization in many cases. We refer to these methods as anonymization methods in the following.

Energy providers can apply such methods to protect user privacy. However, using them has a significant impact on different use cases, as the functioning of local energy markets may have additional costs [BKJB13]. The impact of anonymization on changes in the data is not yet known. A provider intending to anonymize the energy consumption of users nevertheless needs to quantify this effect. This is because the data should remain useful for subsequent analyses. We refer to this scenario as the “anonymization scenario”.

Problem Domain

The result of anonymization using the above-mentioned methods [PLKY07] depends on the basis chosen for the transformation of the data (e.g., Fourier or Wavelet). We thus need to determine how the choice of the basis affects the changes in the data. The other important parameter in this case is the magnitude of the noise σ added to the original data. We conjecture that, the larger the noise added to the data, the larger is the potential impact on the changes.

Setting

Here, the provider considers the general case of data publishing, implying that he has little or no information on the subsequent use of the data. He does not differentiate between a shift in time or importance of changes. He does the same for missed and false-positive changes.

6.1.4 Measure Requirements

The first two contributions of this dissertation concern lossy transformation techniques, such as lossy time-series compression and estimation. The application scenarios above involve the use of such techniques and motivate the need for a measure of their impact on subsequent change detection. Based on these scenarios, we have compiled the following requirements for our measure:

R1: Generalizability The measure should be independent of the change-detection algorithm and should provide meaningful results for any combination of general-purpose change-detection approach and lossy transformation.

R2: Flexibility The user should be able to configure the measure to distinguish and weight four cases according to the subsequent application: shifts of changes in time, modifications of their importance, disappearance of changes and emergence of new ones.

R3: Robustness The measure should be robust. Here, robustness means that computation should return meaningful results for any parametrization of the measure.

6.2 MILTON

We first describe the basic functioning of MILTON. We then present MILTON and say how we have parameterized it.

6.2.1 Problem Definition

A change-detection algorithm *CD* transforms a time series of measurements X into one of events (changes) *CD(X)* $= \{(t_1, s_1), (t_2, s_2), \ldots, (t_m, s_m)\}$. Here, t_i with $i = 1, \ldots, m$ denotes the time the change occurred at, while s_i denotes its score. Many state-of-the-art change-detection approaches associate with each change a score [LYCS13], which characterizes its significance. Without loss of generality, we assume that a change has a score of 1 if a change-detection approach does not provide scores. A lossy transformation T on X produces a modified time series of measurements *T(X)*. Applying *CD* on *T(X)* thus produces a time series of changes *CD(T(X))*, which is possibly different from *CD(X)*. Table 6.1 sums up our notation introduced so far.

Symbol	Definition
X	original time series
T	lossy transformation
CD	change detection algorithm
CD(X)	time series of changes

Table 6.1: Notation Summary

When comparing the changes in *CD(X)* and *CD(T(X))*, we can assign each change to one of the following sets:

$\mathbf{PC} = \mathbf{pairing}(\mathbf{CD}(\mathbf{X}), \mathbf{CD}(\mathbf{T}(\mathbf{X})))$**:** As a result of the lossy transformation of X, changes of *CD(X)* might have been shifted in time or have their score altered. The set *PC* ("paired changes") contains pairs of changes of the form ($x \in$ *CD(X)*, $y \in$ *CD(T(X))*). Here, x is a change of *CD(X)*, and y is its corresponding change in *CD(T(X))*, eventually shifted or of altered score. The **pairing** function identifies and pairs such changes from *CD(X)* and *CD(T(X))*.

$\mathbf{MISS} = \mathbf{CD}(\mathbf{X}) - \mathbf{PC}$**:** As a result of the transformation, some changes of *CD(X)* might not have a match in *CD(T(X))*. The set *MISS* contains such changes, which we call "misses".

$\mathbf{FP} = \mathbf{CD}(\mathbf{T}(\mathbf{X})) - \mathbf{PC}$**:** In contrast, new changes might appear in *CD(T(X))*, which do not have any match in *CD(X)*. We refer to such changes as "false positives", which we add to the set *FP*.

From requirement R2 (cf. Subsection 6.1.4), it follows that MILTON must consider each set defined above differently. In particular, we must determine the changes the transformation has affected in a minor way (minor = shift in time or modification of score; *PC*), how many have disappeared (*MISS*), and how many have emerged as a result of the transformation (*FP*). Second, we should allow setting application-dependent weights on the impact of changes in each set. For example, if missed changes are critical to the subsequent application, our measure must attribute larger weights to such cases than to false positives or vice-versa. To evaluate the impact of a lossy transformation on subsequent change detection, MILTON quantifies how similar *CD(X)* and *CD(T(X))* are, i.e., we sum the weighted differences between the changes *CD(X)* and *CD(T(X))* assigned to *PC*, and the weights of the changes in *MISS* and *FP*.

6.2.2 Calculating PC, MISS and FP

We first explain how the function **pairing**() can be implemented to obtain the set *PC*. Obtaining sets *FP* and *MISS* follows in a straightforward manner.

Finding the optimal matching between changes from *CD(X)* and *CD(T(X))* is not trivial. One reason is that matching two changes can affect the matching of other changes. As an example, suppose that we match two changes $x \in$ *CD(X)* and $y \in$ *CD(T(X))* incorrectly. This means that the correct match y' for x may now be matched with another change incorrectly. This holds for y and its match x' and may cascade. The incorrect matching of two changes may thus impact the entire matching process. The matching process therefore needs to consider all possible matching combinations of changes. Another reason is that the optimal matching may not include all changes in *CD(X)* or *CD(T(X))*, as there may be new changes (false positives) and removed ones (misses). The matching process may therefore need to skip changes. However, it is not clear how many changes it should skip.

Due to Requirement R2, our measure must take into account both the difference in time and score (importance) between two changes. Moreover, it should be possible to weight these differences depending on the application scenario. For example, in the compression scenario (Section 6.1.1), the difference in time has a higher weight than the difference in score. We therefore first define these weights: Let $x = (t_x, s_x)$ be a change of *CD(X)* and $y = (t_y, s_y)$ one of *CD(T(X))*. We define $f_{TIME} : \mathbb{R} \mapsto \mathbb{R}^+$ as a function of the normalized difference in time (Δ_t) between x and y and $f_{SCORE} : \mathbb{R} \mapsto \mathbb{R}^+$ as a function of the normalized difference in score (Δ_s) between x and y. These functions are application-dependent, as explained above. The distance between two changes then is a function g of f_{TIME} and f_{SCORE}:

$$\textit{dist(x, y)} = g(f_{TIME}(\Delta_t(x,y)), f_{SCORE}(\Delta_s(x,y))) \tag{6.1}$$

We fix g as the sum of the contributions f_{MISS} and f_{SCORE}:

$$\textit{dist(x, y)} = f_{TIME}(\Delta_t(x,y)) + f_{SCORE}(\Delta_s(x,y)) \tag{6.2}$$

In our experiments, we have tested other distances, such as the maximum between the two contributions: $max(f_{TIME}(\Delta_t(x,y)), f_{SCORE}(\Delta_s(x,y)))$. This has not lead to substantially different results.

Based on the above-defined distance, one trivial way to find the correspondence between changes *CD(X)* and *CD(T(X))* is to calculate all possible one-to-one combinations of events which maintain the original succession of changes and choose the one with the smallest total distance. However, this is computationally expensive; the number of such combinations grows exponentially with the number of changes in *CD(X)* and *CD(T(X))*. At first sight, the setting may resemble the one of the well-known Hungarian Algorithm. The difference however is the need to maintain the original order of changes for the matching. Several publications however have studied this specific problem or closely-related ones [LWKTM07, WP90]. We use the Optimal Subsequence Bijection (OSB) algorithm introduced in [LWKTM07], which fulfills the matching prerequisites:

- the matching should consider all distances between matched changes
- the matching should allow leaving unmatched changes (misses and false-positives)

However, OSB as is does not solve our problem. This is because, after performing the matching, it does not take unmatched changes into account. OSB matches two sequences *CD(X)* and *CD(T(X))* of (possibly different) lengths m and n:

$$\textit{CD(X)} = \{(t_{x_1}, s_{x_1}), (t_{x_2}, s_{x_2}), \ldots, (t_{x_1}, s_{x_m})\}$$

$$\textit{CD(T(X))} = \{(t_{y_1}, s_{y_1}), (t_{y_2}, s_{y_2}), \ldots, (t_{y_n}, s_{y_n})\}$$

Its goal is to find best-matching subsequences *CD(X)*$'$ of *CD(X)* and *CD(T(X))*$'$ of *CD(T(X))*. Thus, it may skip changes. The authors of OSB motivate having unmatched changes by the fact that the sequences may contain outliers that should be skipped. In our case, these correspond to false-positive and missed changes. However, skipping too much may result in random matches. To avoid this, OSB uses a penalty C for skipping.

The algorithm requires the two sequences, a distance measure and a penalty for skipping changes as input. To find the optimal matching, OSB minimizes the sum of the distances between matched changes and the penalties for changes skipped. It thus considers all distances between matched changes, as required. [LWKTM07] uses the Euclidean distance. We adapt OSB to our case by using the distance from Equation (6.2), next to some other adaptations:

There are many possibilities to set the penalty C. From our experiments, we have found that the standard penalty recommended in [LWKTM07] produces correct matchings and rarely results in mismatches between changes. We therefore used the standard penalty, which is defined as follows:

$$\mathit{C(CD(X), CD(T(X)))} = \operatorname*{mean}_i(\min_j(\mathit{dist}(x_i, y_j)) + \operatorname*{std}_i(\min_j(\mathit{dist}(x_i, y_j))$$

where $x_i \in \mathit{CD(X)}$, $i = 1, \ldots, m$ and $y_j \in \mathit{CD(T(X))}$, $j = 1, \ldots, n$.

Algorithm 4 Algorithm computing *PC*, *MISS* and *FP*

```
1: Let $MISS = \{\}$
2: Let $FP = \{\}$
3: $PC = OSB(CD(X), CD(T(X)), C)$
4: for $x \in CD(X)$ do
5:     if $x \notin PC$ then
6:         $MISS = MISS \cup x$
7:     end if
8: end for
9: for $x \in CD(T(X))$ do
10:    if $x \notin PC$ then
11:        $FP = FP \cup x$
12:    end if
13: end for
```

In our case, OSB outputs a one-to-one pairing between changes in *CD(X)* and *CD(T(X))*, which makes up the set *PC*. To identify the changes which disappeared as a result of the transformation (*MISS*), we loop over changes in *CD(X)* and select those which do not have a match in *CD(T(X))*, i.e., are not in *PC*. Similarly, to obtain new changes (*FP*) we loop over the changes in *CD(T(X))* and select those without a match in *CD(X)*. See Algorithm 4.

6.2.3 Measure Definition

As explained in the previous subsection, having sets *PC*, *MISS* and *FP*, we can construct a general-purpose measure, which satisfies Requirements R1, R2 and R3. We first consider the impact of each set separately, followed by the total impact.

Paired Changes Changes in such a couple may differ in the time when they occur and in their score. As just explained, we quantify this difference using the distance defined in Equation (6.2). To quantify the global impact of such changes, we sum up the distances between "paired changes". This creates the following term, which is part of our measure:

$$errPC = \sum_{(x,y)\in PC} dist(x,y) =$$

$$\sum_{(x,y)\in PC} f_{TIME}(\Delta_t(x,y)) + f_{SCORE}(\Delta_s(x,y))$$

The intuition is that, the more changes are shifted in time and score, the bigger the impact of the transformation on them and vice versa. In case information on the particular impact of shifts in time and score was necessary, $errPC$ could be split into two terms calculated separately:

$$errPC = errTIME + errSCORE$$

where

$$errTIME = \sum_{(x,y)\in PC} f_{TIME}(\Delta_t(x,y))$$

and

$$errSCORE = \sum_{(x,y)\in PC} f_{SCORE}(\Delta_s(x,y))$$

Misses Depending on the application scenario considered, we may want to deal with misses in a differentiated manner according to their score. For example, we may choose to completely ignore missed changes with a low score and conversely assign a bigger weight to ones with a high score. We therefore introduce a weighting function on missed changes f_{MISS} which depends on their score. We discuss how we define this function in the following subsection. To quantify the total impact of missed changes, we sum up their individual impacts weighted by f_{MISS}, yielding the second term of our measure:

$$errMISS = \sum_{(t,s)\in MISS} f_{MISS}(s) \tag{6.3}$$

False Positives As in the case of missed changes, we may want to handle false positives in a differentiated manner depending on their score. For this, we introduce a weighting function on false positives f_{FP}. As in the previous cases, we sum up the individual impacts weighted by f_{FP} and create the last term of our measure:

$$errFP = \sum_{(t,s) \in FP} f_{FP}(s) \tag{6.4}$$

Weight function	Argument
f_{TIME}	shift in time
f_{SCORE}	shift in score
f_{MISS}	missed changes
f_{FP}	false-positive changes

Table 6.2: Notation Summary

Total Impact To quantify the total impact of the different terms introduced above, MILTON sums them up. However, there is another issue, which MILTON should take into account, namely, the number of changes in the original time series $|CD(X)| = |PC| + |MISS|$. We explain the rationale using an example:

Example 5. *Suppose that, for a time series X_1, CD detects 2 changes, while for another time series X_2, it detects 100. Let us further assume that applying transformation T on X_1 introduces a shift in time and score to the original changes $CD(X_1)$ and the summed-up impact is equal to 0.5. We also assume that applying the same transformation T on X_2 leaves all but two changes intact and introduces a shift in time and score to the two changes resulting in an equivalent impact equal to 0.5. Logically, the global impact between the two cases should be significantly different. This is because in the case of X_2, T leaves 98% of the changes intact, while it affects 100% of the changes in the case of X_1. We therefore need to normalize the total impact and divide it by the number of changes CD detects in the original time series.*

Using the above results, we define MILTON as follows:

$$MILTON(X, T, CD) = \frac{errPC + errMISS + errFP}{|PC| + |MISS| + 1} \tag{6.5}$$

We add 1 to the denominator to account for the case when PC and $MISS$ are both empty.

Table 6.2 lists the functions presented above. We have explained the need to use weights within MILTON, and we have provided some intuition on how to set them.

6.2.4 Parametrization

MILTON has four parameters: f_{SCORE}, f_{TIME}, f_{MISS} and f_{FP}. In the following we say how we set these parameters for each use case described in Section 6.1. Observe that the domain of these functions is normalized to the interval $[0, 1]$.

We first consider the compression scenario. Here, we must penalize missed changes substantially more than false positives. We therefore assign a larger weight to f_{MISS} than to f_{FP}. See Table 6.3. Concerning shifts in time and score, we set f_{TIME} proportional to the size of the shift and we set f_{SCORE} equal to zero. This is because we ignore alterations of the importance of changes.

Table 6.3: Measure parametrization

	Compression	Estimation	Anonymization
$f_{TIME}(\Delta_t)$	$\|\Delta_t\|$	$e^{\|\Delta_t\|} - 1$	$\frac{1}{2} \cdot \|\Delta_t\|$
$f_{SCORE}(\Delta_s)$	0	$e^{\|\Delta_s\|} - 1$	$\frac{1}{2} \cdot \|\Delta_s\|$
$f_{MISS}(s)$	$s^2 + 1$	$e^s - 1$	s
$f_{FP}(s)$	s	s	s

We now turn to the estimation scenario. As mentioned, shifts and modifications of the scores are critical to the subsequent application. Thus, we decide to set f_{TIME} and f_{SCORE} to grow exponentially with increasing shifts in time and score (Table 6.3). Regarding f_{MISS} and f_{FP}, we use a similar logic as for the previous scenario where we penalize missed changes substantially more than false-positive ones.

Lastly, we consider the anonymization scenario. As stated, we are in the general case of data publishing. This means that little or no information on the subsequent use of the data is available. We therefore use the average of shifts in time and score (importance) between two changes as distance, with no differentiation between the type of shift (Table 6.3). We let the terms *errFP* and *errMISS* due to missed and false positive changes correspond to the sum of the scores of changes in the respective sets (*MISS* and *FP*).

6.3 Evaluation

MILTON operates as intended if it fulfills the requirements from Section 6.1.4. We have covered the flexibility and robustness requirements by design. To cope with generalizability, we have evaluated MILTON using our three scenarios. In the following, we present the datasets used, the setup of our experiments and their results.

6.3.1 Setup

To evaluate MILTON, we use five datasets:

- The **Reference Energy Disaggregation Dataset (REDD)** (Section 3.2)
- The **Smart Home Dataset (Smart)** (Section 3.3)
- The **Server Dataset** (Section 3.5)
- The **Desktop Dataset** (Section 3.6)
- The **Laptop Dataset** (Section 3.7)

For the evaluation of all scenarios, we have used $CUSUM$ [Pag54], an established change-detection method. We have configured it to detect changes of the mean of a data sequence of at least 5% of its range. We set the parameters (weights) of MILTON according to each application scenario (Table 6.3), cf. Subsection 6.2.4. In the following we present the lossy transformation methods used for each scenario.

Compression Scenario

We use compression techniques based on different classes of models:

a) Adaptive Piecewise Constant Approximation (*APCA*) uses constant functions to approximate segments of data of varying length [KCPM01].

b) Piecewise Linear Histogram (*PWLH*) compresses the data in a similar manner as APCA using straight-line functions instead of constant ones [BSS07].

c) Adaptive Polynomial Piecewise Compression (*APP*) corresponds to the algorithm we proposed and described in Section 4. It combines polynomial functions of different degrees to approximate the data piecewisely in an incremental manner [EEKB15].

All of these methods compress the data, such that the maximum deviation between the original and decompressed data under the uniform norm is smaller than an error threshold ϵ.

Estimation Scenario

We use a dynamic and a calibration-based estimator to obtain the energy-consumption estimates, cf. Section 5 ([EBB14a]).

Anonymization Scenario

We use two data perturbation/anonymization methods: one using the Fourier transform and one using the Wavelet one [PLKY07].

6.3.2 Use Case Evaluation

We now present our evaluation of MILTON.

Compression Scenario

In this scenario, the energy provider wants to identify the method which delivers the best compression ratio for a given impact on the changes in the data together with a good parameter set. For this, we have first computed MILTON for all three compression techniques for different values of the threshold ϵ going from 0.2% to 5% of the range of the time series used. Figure 6.1 shows the average results for the Smart Dataset. They are similar to those we obtained with the REDD Dataset. We observe that the measure generally increases with a growing threshold value. This is because some changes disappear as a result of the rougher compression. We also observe that, for large values of ϵ, compression using *APCA* has a worse impact on the subsequent change detection than the other two methods.

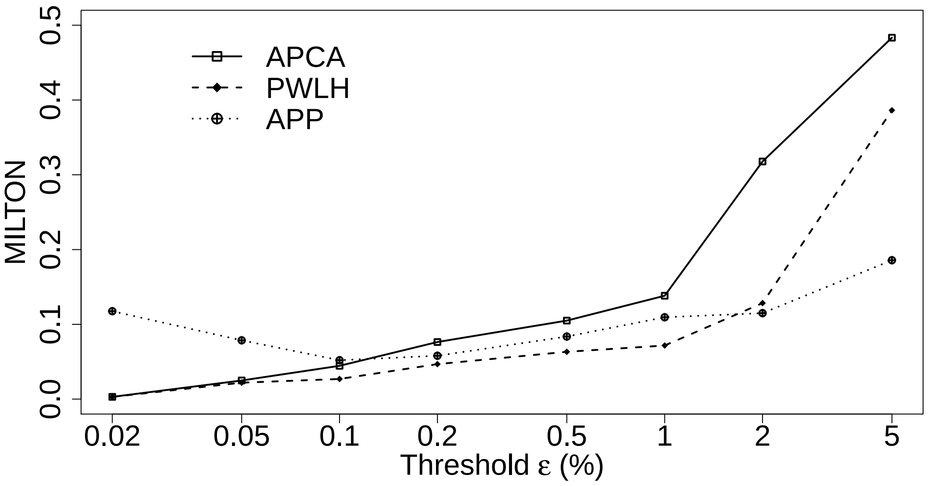


Figure 6.1: MILTON vs. Threshold (ϵ) - Smart Dataset

To understand the reasons behind the results in Figure 6.1, we list and inspect the number of changes in sets *PC*, *MISS* and *FP*, as well as MILTON components *errPC*, *errMISS* and *errFP* for one time series of the Smart dataset (Table 6.4). We first notice that *PWLH* and *APP* preserve existing changes better than *APCA* when $\epsilon > 0.2\%$. This accounts for their lower values of MILTON. Second, in comparison to both *PWLH* and *APCA*, *APP* introduces considerably more false-positive changes. We assume that this is due to the use of polynomials of degree higher than 1. These may introduce "bumps" in the time series, which *CUSUM* interprets as changes. However, they do not impact the value of MILTON significantly, as we set the weight f_{FP} for false-positive changes significantly smaller than for misses (f_{MISS}).

Table 6.4: Number of Changes and Measure Components in Compressed Data for Threshold ϵ – homeB – Smart Dataset

	ϵ	*PC*	*MISS*	*FP*	*errPC*	*errMISS*	*errFP*
APCA	0.02	170	0	0	0.000	0.000	0.000
	0.05	167	3	3	0.006	3.000	0.015
	0.1	160	10	7	0.007	10.000	0.036
	0.2	156	14	7	0.010	14.000	0.038
	0.5	153	17	2	0.007	17.000	0.013
	1	145	25	1	0.010	25.001	0.000
	2	110	60	0	0.008	60.002	0.000
	5	89	81	0	0.000	81.003	0.000
PWLH	0.02	170	0	0	0.000	0.000	0.000
	0.05	164	6	6	0.002	6.000	0.031
	0.1	164	6	6	0.005	6.000	0.031
	0.2	162	8	8	0.006	8.000	0.041
	0.5	160	10	9	0.014	10.000	0.046
	1	157	13	10	0.019	13.000	0.053
	2	148	22	6	0.031	22.001	0.032
	5	110	60	2	0.091	60.002	0.012
APP	0.02	148	22	15	0.013	22.001	0.345
	0.05	147	23	10	0.011	23.001	0.310
	0.1	165	5	4	0.006	5.000	0.021
	0.2	165	5	3	0.022	5.000	0.015
	0.5	159	11	8	0.020	11.000	0.041
	1	154	16	7	0.022	16.000	0.040
	2	154	16	6	0.026	16.000	0.032
	5	138	32	52	0.226	32.001	0.373

We next compute the compression ratio for all three methods for the same val-

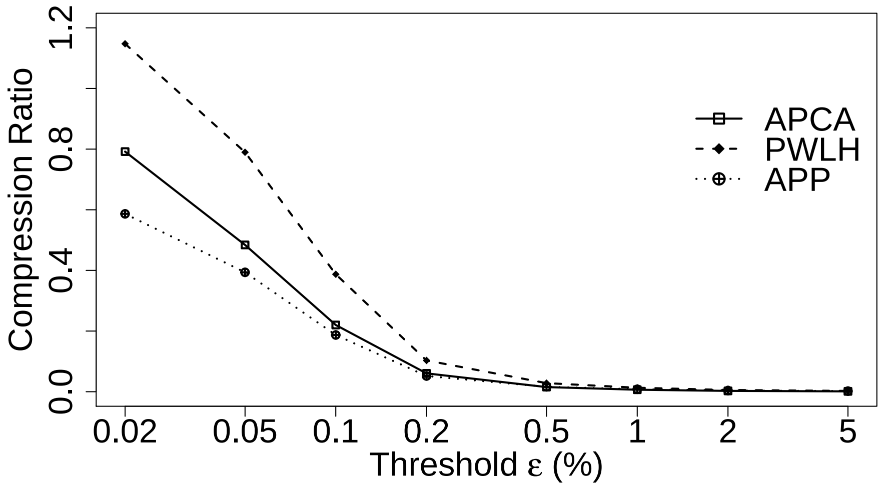


Figure 6.2: Compression Ratio vs. Threshold (ϵ) – Smart Dataset

ues of ϵ as in our previous experiment. We use the following formula:

$$compression\ ratio = \frac{\text{size of compressed data}}{\text{size of initial data}} \tag{6.6}$$

Figure 6.2 shows the average results for the Smart Dataset. We notice that the techniques diverge significantly for small values of ϵ ($\epsilon < 0.2\%$) and converge to similar ones when ϵ grows ($\epsilon > 1\%$). For small values of ϵ, compression with *PWLH* is worst, followed by *APCA* and *APP*.

Using the results above, the provider can identify a good compression method with a good parameter set if a bound on the impact on the changes is given. To this end, he first needs to identify, for each method, the value of ϵ which gives way to an impact within the bound. Then, using the results on compression ratios, he can select the best method.

Summary: *We have shown that MILTON can help the provider decide which compression method suits his needs best. In this use case, for the specific settings used here, no method is generally superior to the other ones.*

Estimation Scenario

In this scenario, a data-center manager wants to identify which estimation method at which aggregation level (e.g., per minute or hourly) has the smallest impact on changes in the data. For this, we aggregate the energy-consumption time series to time intervals from one minute to 60 minutes. We then calculate our measure for both estimators on all datasets for these intervals. Table 6.5 shows the average value of MILTON for time series in each dataset tested. We first notice that the calibration-based estimator has a smaller impact on changes than the dynamic one for almost all datasets and interval lengths. For the laptop dataset, for instance, MILTON is 30% smaller on average. Furthermore, the value of MILTON generally increases with the interval length. This means that, while using a longer time interval for aggregation may improve accuracy as shown in Section 5 ([EBB14a]), it has a bigger impact on changes in the data.

Table 6.5: MILTON by Time Interval Length

		Time interval length (min.)					
		1	2	5	15	30	60
ATIS	Dynamic	0.01	0.01	0.03	0.00	0.00	0.04
	Calibr.	0.00	0.01	0.02	0.00	0.00	0.05
Desktop	Dynamic	0.01	0.03	0.04	0.06	0.11	0.33
	Calibr.	0.00	0.00	0.01	0.01	0.03	0.27
Laptop	Dynamic	0.01	0.01	0.01	0.04	0.06	0.17
	Calibr.	0.00	0.01	0.01	0.02	0.04	0.09

To find out why MILTON behaves in this way in this case, we list the number of changes in the sets *PC*, *MISS* and *FP*, as well as MILTON components *errPC*, *errMISS* and *errFP* for one time series of the Desktop Dataset (Table 6.6). We see that the dynamic estimator usually produces more missed and false positive changes than the calibration-based one. This makes MILTON grow significantly due the definition of f_{MISS} and f_{FP}. Moreover, for small interval lengths the pairing matches changes better than for longer ones. This accounts for the big impact of aggregation on changes for long intervals .

Summary: *For this scenario MILTON lets a data-center manager assess the impact of an estimation method on change detection. The calibration-based estimator impacts changes significantly less than the dynamic one.*

Table 6.6: Number of Changes and Measure Components by Interval Length – Desktop Dataset

	Interval length (min)	*PC*	*MISS*	*FP*	*errPC*	*errMISS*	*errFP*
Dynamic	1	76	37	75	0.614	0.057	0.351
	2	43	1	33	1.191	0.022	0.284
	5	19	1	20	0.472	0.053	0.208
	15	8	0	6	0.431	0.000	0.142
	30	3	0	4	0.312	0.000	0.129
	60	1	2	0	1.055	0.000	0.000
Calibr.-based	1	86	27	0	0.152	0.032	0.012
	2	34	10	1	0.123	0.021	0.003
	5	11	9	0	0.031	0.033	0.000
	15	5	3	0	0.065	0.033	0.000
	30	3	0	0	0.189	0.000	0.000
	60	1	2	0	0.061	0.134	0.000

Anonymization Scenario

In this scenario, an energy provider needs to quantify the impact of anonymization methods on the changes in the data. The goal is to identify the method which protects privacy best while keeping the data useful for subsequent analytics. For this, we computed MILTON for both anonymization methods. We vary the value of the perturbation σ these methods add, as described in [PLKY07].

Figure 6.3 shows the average values of MILTON for House 4 of the REDD dataset when the perturbation σ goes from 0.001 to 50. In this case, these values of σ correspond to the interval $[0.01\%, 58\%]$ of the standard deviation of the energy-consumption time series of House 4. We obtained similar results for all other REDD and Smart time series.

We observe that MILTON increases if we add more perturbation when using the Wavelet transform, while it stays practically constant when using Fourier. To understand why this happens, we show the number of changes in the sets *PC*, *MISS* and *FP*, as well as MILTON components *errPC*, *errMISS* and *errFP* for one time series of House 4 in Table 6.7. The number of paired changes and missed ones varies only slightly in both cases. However, adding a bigger perturbation when using the Wavelet transform introduces more false-positive changes (*FP*), which makes MILTON grow. We believe that this is due to the nature of the Haar-Wavelet, which the Wavelet-based method uses. Adding

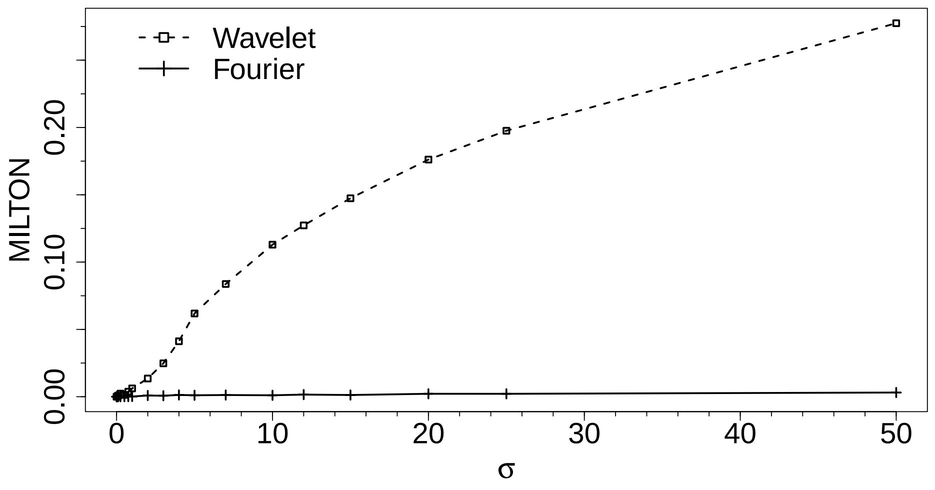


Figure 6.3: Milton vs. perturbation added σ – House 4 – REDD Dataset

perturbation in the form of Haar-Wavelets of significant magnitude introduces changes which have not been present in the data originally.

Table 6.7: Number of Changes and Measure Components by Perturbation σ – House 4 – REDD Dataset

	σ	*PC*	*MISS*	*FP*	*errPC*	*errMISS*	*errFP*
Fourier	1	160	0	0	0.001	0.000	0.000
	2	160	0	0	0.002	0.000	0.000
	5	156	4	4	0.004	0.394	0.394
	10	159	1	1	0.011	0.209	0.209
	25	158	2	3	0.033	0.148	0.168
	50	157	3	5	0.071	0.213	0.257
Wavelet	1	153	7	9	0.025	1.109	1.164
	2	157	3	6	0.065	0.370	0.434
	5	149	11	120	0.044	1.965	4.931
	10	152	8	336	0.048	1.171	11.606
	25	148	12	469	0.038	2.158	26.093
	50	152	8	498	0.047	0.984	38.317

Summary: *Using MILTON, an energy provider can quantify the impact of anonymization methods on subsequent change detection. In this case, the evaluation of the two methods (Fourier-based and Wavelet-based) shows that they have a significantly different impact on the changes. This is even though they protect the privacy on most of the datasets tested to a similar extent according to [PLKY07].*

6.4 Summary

Recent research has proposed numerous lossy transformation techniques for time-series data. While transforming the data, they may impact characteristics of the data, such as changes, which are important for further analyses. To address this issue, we have developed a generalizable and flexible measure which quantifies the impact of a lossy transformation on subsequent change detection. Our evaluation shows that it is useful for various application scenarios to identify adequate parameters of a lossy transformation so that its advantages are maximized while the impact on subsequent change detection is bounded.

7 Conclusions and Outlook

Fine-granular time-series data enables many useful applications. However, collecting it in high volumes generates several kinds of problems, which we need to address. In this dissertation, we have investigated two kinds of problems caused by the collection of huge volumes of time series. As our solutions include lossy transformations of the data, we furthermore investigated their impact on subsequent data analysis. The following chapter concludes this dissertation by summarizing the most important contributions, the lessons learned, as well as interesting, related subjects for future work.

7.1 Summary

The first part of this dissertation investigated the problem of reducing the storage space for numerical time series. Here, we inspected the relatively recent direction of research in lossy time series compression through piecewise approximation. Recent research has provided several methods which use mathematical functions to piecewisely approximate a time series. The approximating functions are subsequently stored instead of the original points. The performance of these approaches depends strongly on the nature of the data as most approaches use a single class of mathematical functions, e.g., constant or straight-line functions, for their approximations. We introduced the idea of combining multiple functions in an incremental manner with a new lossy compression method. Our method employs polynomials of up to a certain degree, which is a parameter the user needs to set. We therefore next investigated the problem of determining this parameter for a given dataset. Here we developed two methods of estimating the compression ratio of our method for different values of this parameter. We have evaluated both our compression method and our methods for setting its parameters using publicly available real-world datasets.

The second part of our dissertation considered another challenge when collecting time-series data. Such data is usually measured and communicated by specific sensors (e.g., smart electricity meters), which need to be installed. This is thus in large scales expensive. We have therefore researched another way of obtaining this data – by estimating the values, instead of measuring them directly. We have investigated the case of estimating computer energy consumption, for which recent research provided many methods of estimation. To address these challenges, we first investigated application scenarios which use estimates of computer energy consumption. We explicitely studied scenarios that have different requirements in terms of estimation effort and requirements on the accuracy of the estimates. Using these scenarios, we assembled requirements on a general framework for estimating computer energy consumption. We next developed three generalized estimation methods to form FRESCO, A FRamework for the Energy eStimation of COmputers. FRESCO integrates these methods into an estimation workflow which covers a wide range of accuracy requirements and computer systems.

The first and second part of this dissertation included methods which provide a transformed version of the original time series. In the first part, this corresponds to the lossy compression method. The second part includes methods which provide time series of estimates. The third part of our dissertation investigated the impact of such transformation methods on subsequent data analysis. This is important because the transformed time series needs to remain useful fur subsequent analytics. The challenges for determining this impact are manifold, as such an impact can, for instance, modify, shift or remove changes from time series. Our goal was to design a a generally applicable measure for quantifying this impact. Our work started with an investigation of application scenarios which involve different transformation techniques. This was necessary in order gave way to the requirements on the measure. We then presented MILTON, a Measure which quantifies the Impact of various Lossy Transformation methods for time series on subsequent change detectiON. We defined the problem we need to solve when calculating MILTON. We subsequently described how we solve this problem and how we calculate MILTON. We then explained its parameters and how to set them for different application scenarios. We finally evaluated our measure using three application scenarios and real-world datasets.

7.1.1 Lessons Learned

We have experienced and learned many things while working on this dissertation. In the following, we point out the most important lessons we have learned.

Choice of norm for error calculation is essential. The Euclidean (or L_2 norm) is often used to approximate and segment time-series. Its main advantage is that the function calculating the error is differentiable and many established optimization algorithms can be used (e.g., gradient descent). However, it does not offer a guarantee on the approximation, such as with the uniform norm (also called maximum or L_∞ norm), which we used within our approach for compressing time-series. The use of the uniform norm allows us to have a guarantee on the deviation between any corresponding approximated and original values. Nonetheless, this comes at a cost since the approximation becomes complex. This is because the error function is non-differentiable due to the use of the absolute value function. Thus, the choice of the norm for the error calculation is crucial for the choice of optimal or adequate algorithms for the piecewise approximation of time-series.

Energy consumption data poses multiple challenges. We have learned that the Smart Grid is an interesting context for research due to several reasons. One important reason is that many challenges occur simultaneously when collecting energy consumption data, which do not occur in other domains. Thus, collecting such data involves (i) big volumes of storage space, (ii) big effort of capture using sensor nodes and (iii) easy access to private information. To our knowledge, these challenges rarely occur simultaneously in other domains. For example, in the context of financial time-series (e.g., stock quote data), there is little need to use privacy preserving methods or installing specific sensors to capture the data.

Availability of real-world datasets is crucial. Even if this seems obvious, we would like to emphasize the importance of availability of public real-world datasets. In this dissertation, we have used several real-world datasets for the evaluation of our approaches. Datasets, such as REDD or Smart* (Section 3), enabled us to perform a thorough investigation of our contributions and their comparison with related work. On the other hand, for other domains, such as change-detection, we noted a lack of corresponding real-world labeled datasets. This makes the investigation of the domain particularly difficult. Many change-detection approaches we have studied are thus evaluated using mostly synthetic datasets and a comparison with related work is missing or incomplete.

7.2 Future Work

The work in this dissertation can be extended with a number of interesting research questions, which we did not address to limit its scope. We conclude this dissertation with several such research topics, which are built on or related to it.

7.2.1 Improvement of Lossy Piecewise Compression by Partial Presorting

As mentioned in the motivation of our work, piecewise compression methods mostly use a single type of mathematical functions for the approximation, e.g., constant or straight-line functions. Our idea was to combine multiple types in an incremental manner, such that our method can also adapt to variable data. Another way to deal with the variable parts of time series would be to presort these and only to compress them subsequently [LSE^{+}11]. This comes at a cost. First, the data needs to be sorted. Second, the arrangement of the original values needs to be stored as well. Third, to access to an individual value of the time series, we need to first find the place of the value in the arrangement and then calculate it. Thus, this solution should be employed only when the data is too variable to be handled by approximation with a high-order polynomial.

7.2.2 Extension of Energy Consumption Estimation to General-Purpose Electrical Devices

The Internet of Things (IoT) is steadily becoming a reality. This enables more and more objects, among which general-purpose electrical devices, to communicate over the Internet. An extension of our FRESCO, our framework for computer energy estimation, to general-purpose electrical devices becomes therefore interesting. Thus, instead of installing a smart electricity meter for every device in a household, it could be possible to obtain estimates using general information and existing sensors on a given device. Let us consider a washing machine as an example for an electrical device. For a rough estimate of its consumption, general information on its power consumption may be used. On the other hand, for a more accurate estimate, it could be possible to perform a profiling of the different cycles it can execute and estimate its energy consumption based on that profile. As in our work, the extension of FRESCO for such devices implies the investigation of application scenarios with different requirements on the accuracy of the estimates and effort invested to obtain them.

7.2.3 Parameter Learning for Change Detection

In the course of our investigation of state-of-the-art methods for change detection, we discovered several interesting facts. First, many methods have several parameters which must be set, e.g., length of window under consideration, detection threshold. Second, in many cases, there is little or no information available on how to set these parameters. One important fact here is that these parameters depend strongly on the dataset on which change detection takes place. One way to identify good values for these parameters is to perform thorough empirical tests, which some of the authors of the algorithms claim to have done. Another way would be to *learn* the values by using ground truth data. The idea is thus to employ an optimization method to find good parameter values using annotated ground-truth data. In principle, any optimization method may be used to learn the parameters. Furthermore, it is possible to investigate incremental methods, such that the learning occurs as more and more annotated data is available. Additionally, as few ground-truth datasets are available, such data is difficult to obtain. Thus, making general ground-truth datasets publicly available may boost research in the field of change detection. Last but not least, it would be possible to apply the idea of ensembling (as in the case of classification) to combine several change detection techniques and boost the overall performance.

7.2.4 A Framework for Time-Series Forecasting

We used a general state-of-the-art forecasting method in one of the scenarios of the evaluation of our lossy compression method. Furthermore, time series forecasting could be used as an interesting subsequent application after applying other lossy transformation methods as well. While investigating different forecasting methods, we discovered several interesting facts. First, there are numerous time-series forecasting methods. It is thus difficult to choose an appropriate method for a given application scenario. Second, these methods have different parameters which depend on their subsequent application, e.g., forecast horizon, granularity of forecasts. Third, forecasting methods have different complexities and costs of execution (computation time): ranging from rather simple auto-regressive models to multi-layer neural networks. Fourth, several benchmark studies of state-of-the-art forecasting methods have shown that there is no clear winner among them [VGT$^+$14].

It may thus be interesting to develop a general-purpose framework for time-series forecasting. The framework would take several inputs: historical and other external data, parameters of the forecast (horizon, etc.), requirements on the accuracy and effort to be invested (complexity of the forecasting method).

Based on this, it would then suggest one or more appropriate forecasting methods, which it would subsequently instantiate. A first step towards developing this framework is a thorough investigation of existing state-of-the-art forecasting methods, as well as determining the requirements for the framework. To simplify the problem in the first instance one could first create an extendable framework, which would be able to integrate further forecasting methods using a small effort.

To conclude this dissertation, we have addressed several challenges related to the collection of high volumes of time-series data and we have shown that our solutions are useful. Given the directions for future work above, we believe that other research fields dealing with time-series data will benefit from this dissertation and that this area remains an exciting field of research.

Appendix

.1 Appendix

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

.1.1 Proof for Lower 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 4.3.2, we first show that our lower-bound model is equivalent to the $swing$ filter introduced in [EEC+09]. We then use the following fact from [EEC+09]: 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), ..., $(t_i, X_i)]$, 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 $(t_2, X_2 + \epsilon)$, ..., $(t_i, X_i + \epsilon)$. 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 $(t_2, X_2 - \epsilon)$, ..., $(t_i, X_i - \epsilon)$. 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:

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

The equivalent condition for our lower bound model (Condition 4.9) 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 [EEC+09]: 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.

.1.2 Proof for Upper Bound

Proof: To prove Lemma 4.3.3, we use Lemma 4.1 for univariate time series from [EEC+09]:

Lemma .1.1. *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 $(t_h, X_h - \epsilon)$ and $(t_l, X_l + \epsilon)$ $((t_h, X_h + \epsilon)$ and $(t_l, X_l - \epsilon))$, 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 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 4.14 holds as well.

List of Figures

List of Tables

Bibliography

[AC11] Gergely Acs and Claude Castelluccia. I have a dream! (differentially private smart metering). In *Information Hiding*, 2011.

[AM07] Ryan Prescott Adams and David J. C. MacKay. Bayesian online changepoint detection. *arXiv preprint arXiv:0710.3742*, 2007.

[ASWW05] Reza Azimi, Michael Stumm, and Robert W. Wisniewski. Online performance analysis by statistical sampling of microprocessor performance counters. In *Annual International Conference on Supercomputing*, 2005.

[BBBK13] Erik Buchmann, Klemens Böhm, Thorben Burghardt, and Stephan Kessler. Re-identification of smart meter data. *Personal and Ubiquitous Computing*, 2013.

[BC94] Donald J. Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *KDD workshop*, 1994.

[BD02] Peter J. Brockwell and Richard A. Davis. *Introduction to time series and forecasting*. Springer, 2002.

[BdM12] Robert Basmadjian and Hermann de Meer. Evaluating and modeling power consumption of multi-core processors. In *International Conference on Future Energy Systems (e-Energy)*, 2012.

[Bel00] Frank Bellosa. The benefits of event-driven energy accounting in power-sensitive systems. In *ACM SIGOPS European Workshop*, 2000.

[BG] Albert Bifet and Ricard Gavald. *Learning from Time-Changing Data with Adaptive Windowing*, chapter 42.

[BGM$^+$12] Ramon Bertran, Marc Gonzalez, Xavier Martorell, Nacho Navarro, and Eduard Ayguad. A systematic methodology to generate decomposable and responsive power models for cmps. *IEEE Transactions on Computers*, 2012.

[BJ76] George E. P. Box and Gwilym M. Jenkins. *Time series analysis: forecasting and control*. Holden-Day, 1976.

[BJ12] William Lloyd Bircher and Lizy K. John. Complete system power estimation using processor performance events. *IEEE Transactions on Computers*, 61:563–577, 2012.

[BKJB13] Erik Buchmann, Stephan Kessler, Patrick Jochem, and Klemens Böhm. The costs of privacy in local energy markets. In *IEEE Conference on Business Informatics*, 2013.

[BKS12] Birger Becker, Anna Kellerer, and Hartmut Schmeck. User interaction interface for energy management in smart homes. In *IEEE PES Innovative Smart Grid Technologies (ISGT)*, 2012.

[BLFdM13] Andreas Berl, Gergo Lovasz, von Tüllenburg Ferdinand, and Hermann de Meer. Modelling power adaption flexibility of data centres for demand-response management. In *Energy Efficiency in Large Scale Distributed Systems*. 2013.

[BMI$^+$12] Sean Barker, Aditya Mishra, David Irwin, Emmanuel Cecchet, Prashant Shenoy, and Jeannie Albrecht. Smart*: an open data set and tools for enabling research in sustainable homes. *SustKDD Workshop on Data Mining Applications in Sustainability*, 2012.

[BNdM12] Robert Basmadjian, Florian Niedermeier, and Hermann de Meer. Modelling and analysing the power consumption of idle servers. In *Sustainable Internet and ICT for Sustainability (SustainIT)*, 2012.

[BSS07] Chiranjeeb Buragohain, Nisheeth Shrivastava, and Subhash Suri. Space efficient streaming algorithms for the maximum error histogram. In *IEEE International Conference on Data Engineering*, 2007.

[Bun10] Bundesregierung Deutschland [German Federal Government]. Energiekonzept für eine umweltschonende, zuverlässige und bezahlbare Energieversorgung [Energy concept for an environmentally-friendly, reliable, and affordable energy supply], September 2010.

[BVLKJ05] William Lloyd Bircher, Madhavi Valluri, Jason Law, and Lizy K. John. Runtime identification of microprocessor energy saving opportunities. In *International Symposium on Low Power Electronics and Design*, 2005.

[BVN93] Michèle Basseville and Igor V. Nikiforov. *Detection of abrupt changes: theory and application*. Prentice Hall Englewood Cliffs, 1993.

[CF99] Kin-Pong Chan and Ada Wai-chee Fu. Efficient time series matching by wavelets. In *IEEE International Conference on Data Engineering*, 1999.

[CLS12] Hui Chen, Youhuizi Li, and Weisong Shi. Fine-grained power management using process-level profiling. *Sustainable Computing: Informatics and Systems*, 2012.

[CN04] Yuhan Cai and Raymond Ng. Indexing spatio-temporal trajectories with chebyshev polynomials. In *ACM SIGMOD International Conference on Management of Data*, 2004.

[CS03] Remy Cottet and Michael Smith. Bayesian modeling and forecasting of intraday electricity load. *Journal of the American Statistical Association*, 2003.

[CS11] Hui Chen and Weisong Shi. *Handbook on Energy-Aware and Green Computing*, chapter Power Measurement and Profiling: State-of-the-Art. 2011.

[Dar06] Sarah Darby. The effectiveness of feedback on energy consumption: a review for DEFRA of the liternature on metering, billing and direct displays. Technical report, Environmental Change Institute, University of Oxford, 2006.

[dat10] Private dataset. Campus dataset. 2010.

[DBF$^+$10] Lars Dannecker, Matthias Boehm, Ulrike Fischer, Frank Rosenthal, Gregor Hackenbroich, and Wolfgang Lehner. State-of-the-art report on forecasting – a survey of forecast models for energy demand and supply. Technical report, The MIRACLE Consortium, 2010.

[DDD05] Frdric Desobry, Manuel Davy, and Christian Doncarli. An online kernel change detection algorithm. *IEEE Transactions on Signal Processing*, 2005.

[DL06] Marco Dalai and Riccardo Leonardi. Approximations of one-dimensional digital signals under the l^∞ norm. *IEEE Transactions on Signal Processing*, 2006.

[DMR10] Gaurav Dhiman, Kresimir Mihic, and Tajana Rosing. A system for online power prediction in virtualized environments using gaussian mixture models. In *Design Automation Conference*, 2010.

[DRS09] Thanh Do, Suhib Rawshdeh, and Weisong Shi. ptop: A process-level power profiling tool. In *HotPower*, 2009.

[Dwo06] Cynthia Dwork. Differential privacy. In *Automata, languages and programming*. Springer, 2006.

[EBB14a] Pavel Efros, Erik Buchmann, and Klemens Böhm. FRESCO: a framework to estimate the energy consumption of computers. In *IEEE Conference on Business Informatics*, 2014.

[EBB14b] Pavel Efros, P., Erik Buchmann, and Klemens Böhm. Fresco: A framework to estimate the energy consumption of computers. *Karlsruhe Reports in Informatics*, 4, 2014. Technical Report.

[EBEB15] Pavel Efros, Erik Buchmann, Adrian Englhardt, and Klemens Böhm. How to quantify the impact of lossy transformations on change detection. In *International Conference on Scientific and Statistical Database Management (SSDBM)*, 2015.

[EEC$^+$09] Hazem Elmeleegy, Ahmed K. Elmagarmid, Emmanuel Cecchet, Walid G. Aref, and Willy Zwaenepoel. Online piece-wise linear approximation of numerical streams with precision guarantees. In *International Conference on Very Large Data Bases (VLDB)*, 2009.

[EEKB15] Frank Eichinger, Pavel Efros, Stamatis Karnouskos, and Klemens Böhm. A time-series compression technique and its application to the smart grid. *The VLDB Journal*, 24:193–218, 2015.

[EMDL10] Karen Ehrhardt-Martinez, Kat A. Donnelly, and John A. Laitner. Advanced metering initiatives and residential feedback programs: a meta-review for household electricity-saving opportunities. Technical report, American Council for an Energy-Efficient Economy, 2010.

[EPVM12] Frank Eichinger, Daniel Pathmaperuma, Harald Vogt, and Emmanuel Müller. Data analysis challenges in the future energy domain. *Computational Intelligent Data Analysis for Sustainable Development*, 2012.

[ERK06] Dimitris Economou, Suzanne Rivoire, and Christos Kozyrakis. Full-system power analysis and modeling for server environments. In *Workshop on Modeling Benchmarking and Simulation (MOBS)*, 2006.

[Fel51] William Feller. The asymptotic distribution of the range of sums of independent random variables. *The Annals of Mathematical Statistics*, 1951.

[FH12] Shu Fan and Rob J. Hyndman. Short-term load forecasting based on a semi-parametric additive model. *IEEE Transactions on Power Systems*, 2012.

[Fra12] Fraunhofer-Institut für Windenergie und Energiesystemtechnik (IWES), Ingenieurbüro für neue Energien (IfnE), Deutsches Zentrum für Luft- und Raumfahrt (DLR). Langfristszenarien und Strategien für den Ausbau der erneuerbaren Energien in Deutschland bei Berücksichtigung der Entwicklung in Europa und global [Long-term scenarios and strategies for the development of renewable energy in Germany considering development in Europe and globally]. Technical report, Expert Report for the Federal Ministry for the Environment, Nature Conservation and Nuclear Safety, 2012.

[FRM94] Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. Fast subsequence matching in time-series databases. *SIGMOD Record*, 1994.

[FWB07] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. In *Annual International Symposium on Computer Architecture*, 2007.

[FWCY10] Benjamin C. M. Fung, Ke Wang, Rui Chen, and Philip S. Yu. Privacy-preserving data publishing: A survey of recent developments. *ACM Computing Surveys*, 2010.

[GCG$^+$13] William Gu, Jaesik Choi, Ming Gu, Horst Simon, and Kesheng Wu. Fast change point detection for electricity market analysis. In *IEEE International Conference on Big Data*, 2013.

[GFS$^+$10] Rong Ge, Xizhou Feng, Shuaiwen Song, Hung-Ching Chang, Dong Li, and Kirk W. Cameron. Powerpack: Energy profiling and analysis of high-performance systems and applications. *IEEE Transactions on Parallel and Distributed Systems*, 2010.

[Gmb06] SWKiel Netz GmbH. VDEW-Lastprofile, 2006. Accessed 25 April 2013.

[GS99] Valery Guralnik and Jaideep Srivastava. Event detection from time series data. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1999.

[HDVC$^+$05] Taliver Heath, Bruno Diniz, Enrique V. Carrera, Wagner Meira Jr, and Ricardo Bianchini. Energy conservation in heterogeneous server clusters. In *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2005.

[HJA13] Nguyen Quoc Viet Hung, Hoyoung Jeung, and Karl Aberer. An evaluation of model-based approaches to sensor data compression. *IEEE Transactions on Knowledge and Data Engineering*, 2013.

[HK06] Rob J. Hyndman and Anne B. Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, 2006.

[Huf52] David A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the Institute of Radio Engineers*, 1952.

[IKG12] Dejan Ilic, Stamatis Karnouskos, and Per Goncalves Da Silva. Sensing in power distribution networks via large numbers of smart meters. In *Conference on Innovative Smart Grid Technologies (ISGT)*, 2012.

[IM03] Canturk Isci and Margaret Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *IEEE/ACM International Symposium on Microarchitecture*, 2003.

[Int] Intel. Intelligent platform management interface. http://www.intel.com/design/servers/ipmi/index.htm.

[Jan01] Jeff Janzen. Calculating memory system power for DDR SDRAM. *Designline*, 2001.

[JCG+11] Victor Jimenez, Francisco Cazorla, Roberto Gioiosa, Eren Kursun, Canturk Isci, Alper Buyuktosunoglu, Pradip Bose, and Mateo Valero. A case for energy-aware accounting and billing in large-scale computing facilities cost metrics and design implications, 2011.

[JGC+11] Victor Jimenez, Roberto Gioiosa, Francisco J. Cazorla, Mateo Valero, Eren Kursun, Canturk Isci, Alper Buyuktosunoglu, and Pradip Bose. Energy-aware accounting and billing in large-scale computing facilities. *IEEE Micro*, 2011.

[JLB10] Andrej Jokić, Mircea Lazar, and Paul P. J. van den Bosch. Price-based control of electrical power systems. In *Intelligent Infrastructures*. Springer, 2010.

[JM01] Russ Joseph and Margaret Martonosi. Run-time power estimation in high performance microprocessors. In *International Symposium on Low Power Electronics and Design*, 2001.

[KAMSK09] Sanjeev Kumar Aggarwal, Lalit Mohan Saini, and Ashwani Kumar. Electricity price forecasting in deregulated markets: a review and evaluation. *International Journal of Electrical Power and Energy Systems*, 2009.

[Kar11] Stamatis Karnouskos. Demand side management via prosumer interactions in a smart city energy marketplace. In *International Conference on Innovative Smart Grid Technologies (ISGT)*, 2011.

[KCPM01] Eamonn Keogh, Kaushik Chakrabarti, Michael Pazzani, and Sharad Mehrotra. Locally adaptive dimensionality reduction for indexing large time series databases. *ACM SIGMOD Record*, 2001.

[KGI11] Stamatis Karnouskos, Per Goncalves Da Silva, and Dejan Ilic. Assessment of high-performance smart metering for the web service enabled smart grid. In *International Conference on Performance Engineering (ICPE)*, 2011.

[KGI12] Stamatis Karnouskos, Per Goncalves Da Silva, and Dejan Ilic. Energy services for the smart grid city. In *International Conference on Digital Ecosystem Technologies – Complex Environment Engineering (DEST-CEE)*, 2012.

[KJ11] J Zico Kolter and Matthew J Johnson. Redd: A public data set for energy disaggregation research. In *SIGKDD Workshop on Data Mining Applications in Sustainability*, 2011.

[KJP98] Eamonn J. Keogh and Michael J. Pazzani. An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1998.

[KK02] Eamonn Keogh and Shruti Kasetty. On the need for time series data mining benchmarks: a survey and empirical demonstration. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2002.

[Koo11] Jonathan Koomey. Growth in data center electricity use 2005 to 2010. Technical report, Analytics Press, 2011.

[KZL$^+$10] Aman Kansal, Feng Zhao, Jie Liu, Nupur Kothari, and Arka A Bhattacharya. Virtual machine power metering and provisioning. In *ACM Symposium on Cloud Computing*, 2010.

[Lau05] James Laudon. Performance/watt: The new server focus. *ACM SIGARCH Computer Architecture News*, 2005.

[LBSB07] Yann-Aël Le Borgne, Silvia Santini, and Gianluca Bontempi. Adaptive model selection for time series prediction in wireless sensor networks. *Signal Processing*, 2007.

[LDR06] Kristen LeFevre, David J DeWitt, and Raghu Ramakrishnan. Workload-aware anonymization. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006.

[LKJ03] Tao Li and Lizy Kurian John. Run-time modeling and estimation of operating system power consumption. *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 2003.

[LKWL07] Jessica Lin, Eamonn Keogh, Li Wei, and Stefano Lonardi. Experiencing SAX: A novel symbolic representation of time series. *Data Mining and Knowledge Discovery*, 2007.

[LM03] Iosif Lazaridis and Sharad Mehrotra. Capturing sensor-generated time series with quality guarantees. In *International Conference on Data Engineering*, 2003.

[LS05] Kyeong-Jae Lee and Kevin Skadron. Using performance counters for runtime temperature sensing in high-performance processors. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2005.

[LSE$^+$11] Sriram Lakshminarasimhan, Neil Shah, Stephane Ethier, Scott Klasky, Rob Latham, Rob Ross, and Nagiza F. Samatova. Compressing the incompressible with isabela: In-situ reduction of spatio-temporal data. In *Euro-Par 2011 Parallel Processing*. Springer Berlin Heidelberg, 2011.

[LWKTM07] Longin J. Latecki, Qiang Wang, Susan Koknar-Tezel, and Vasileios Megalooikonomou. Optimal subsequence bijection. In *IEEE International Conference on Data Mining (ICDM)*, 2007.

[LYCS13] Song Liu, Makoto Yamada, Nigel Collier, and Masashi Sugiyama. Change-point detection in time-series data by relative density-ratio estimation. *Neural Networks*, 2013.

[MB06] Andreas Merkel and Frank Bellosa. Balancing power consumption in multiprocessor systems. In *ACM SIGOPS/EuroSys European Conference on Computer Systems*, 2006.

[MCLRB09] Milan Milenkovic, Enrique Castro-Leon, and James R. Blakley. Power-aware management in cloud data centers. In *International Conference on Cloud Computing (CloudCom)*, 2009.

[MCWJH98] Spyros G. Makridakis, Steven C. Wheelwright, and Rob J. Hyndman. *Forecasting: Methods and Applications*. Wiley, 1998.

[Mit97] Tom Mitchell. *Machine Learning*. McGraw Hill, 1997.

[MMSF+10] Andrés Molina-Markham, Prashant Shenoy, Kevin Fu, Emmanuel Cecchet, and David Irwin. Private memoirs of a smart meter. In *ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Building*, 2010.

[MSW10] Friedemann Mattern, Thorsten Staake, and Markus Weiss. ICT for green: How computers can help us to conserve energy. In *International Conference on Energy-Efficient Computing and Networking (E-Energy)*, 2010.

[NBRS12] Adel Noureddine, Aurelien Bourdon, Romain Rouvoy, and Lionel Seinturier. Runtime monitoring of software energy hotspots. In *IEEE/ACM International Conference on Automated Software Engineering*, 2012.

[NSQ+12] Dao Viet Nga, Ong Han See, Do Nguyet Quang, Chee Yung Xuen, and Lai Lee Chee. Visualization techniques in smart grid. *Smart Grid and Renewable Energy*, 2012.

[oE09] U.S. Department of Energy. President obama calls for greater use of renewable energy. Weekly Electronic Newsletter, January 2009.

[oE13] U.S. Department of Energy. Estimating appliance and home electronic energy use, 2013. Accessed 20 November 2013.

[Pag54] E. S. Page. Continuous inspection schemes. *Biometrika*, 1954.

[Par09] European Parliament. Renewables directive 2009/28/ec, April 2009.

[PD11] Peter Palensky and Dietmar Dietrich. Demand side management: Demand response, intelligent energy systems, and smart loads. *IEEE Transactions on Industrial Informatics*, 2011.

[PLKY07] Spiros Papadimitriou, Feifei Li, George Kollios, and Philip S. Yu. Time series compressibility and privacy. In *International Conference on Very Large Data Bases*, 2007.

[PN11] Meikel Poess and Raghunath Othayoth Nambiar. Power based performance and capacity estimation models for enterprise information systems. *IEEE Data Engineering Bulletin*, 2011.

[PRA11] Thanasis G. Papaioannou, Mehdi Riahi, and Karl Aberer. Towards online multi-model approximation of time series. In *IEEE International Conference on Mobile Data Management*, 2011.

[RASRK07] Suzanne Rivoire, Mehul A. Shah, Parthasarathy Ranganathan, and Christos Kozyrakis. Joulesort: A balanced energy-efficiency benchmark. In *ACM SIGMOD international conference on Management of data*, 2007.

[REG$^+$01] Ramu Ramanathan, Robert Engle, Clive WJ Granger, Farshid Vahid-Araghi, and Casey Brace. *Short-run forecasts of electricity loads and peaks*. Cambridge University Press, 2001.

[REWG$^+$97] Ramu Ramanathan, Robert Engle, Clive W.J. Granger, Farshid Vahid-Araghi, and Casey Brace. Short-run forecasts of electricity loads and peaks. *International Journal of Forecasting*, 1997.

[Riv08] Suzanne Marion Rivoire. *Models and Metrics for Energy-efficient Computer Systems*. PhD thesis, Stanford University, 2008.

[RK05] Chotirat Ann Ratanamahatana and Eamonn Keogh. Three myths about dynamic time warping data mining. In *SIAM International Conference on Data Mining*, 2005.

[RLB13] Goce Ristanoski, Wei Liu, and James Bailey. A time-dependent enhanced support vector machine for time series regression. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2013.

[RLG$^+$10] Chotirat Ann Ratanamahatana, Jessica Lin, Dimitrios Gunopulos, Eamonn Keogh, Michail Vlachos, and Gautam Das. Mining time series data. In *Data Mining and Knowledge Discovery Handbook*. Springer, 2010.

[RRK08] Suzanne Rivoire, Parthasarathy Ranganathan, and Christos Kozyrakis. A comparison of high-level full-system power models. *HotPower*, 8:3–3, 2008.

[RRR$^+$12] Martin Ringwelski, Christian Renner, Andreas Reinhardt, Andreas Weigely, and Volker Turau. The hitchhiker's guide to choosing the compression algorithm for your smart meter data. In *International Conference and Exhibition (ENERGYCON)*, 2012.

[RSMP11] S. Raj Rajagopalan, Lalitha Sankar, Soheil Mohajer, and H. Vincent Poor. Smart meter privacy: A utility-privacy framework. In *IEEE International Conference on Smart Grid Communications*, 2011.

[Sal08] David Salomon. *A Concise Introduction to Data Compression*. Springer, 2008.

[Sei91] Raimund Seidel. Small-dimensional linear programming and convex hulls made easy. *Discrete & Computational Geometry*, 1991.

[SK08] Jin Shieh and Eamonn Keogh. SAX: Indexing and mining terabyte sized time series. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2008.

[SMPH05] David C. Snowdon, Stefan M. Petters, and Gernot Heiser. Power measurement as the basis for power management. In *Workshop on Operating Systems Platforms for Embedded Real-Time applications*, 2005.

[spa] Spam assassin. spamassassin.apache.org. Accessed 1 October 2015.

[STR10] Yunus Saatçi, Ryan D Turner, and Carl E Rasmussen. Gaussian process change point models. In *International Conference on Machine Learning*, 2010.

[STZ00] Cyrus Shahabi, Xiaoming Tian, and Wugang Zhao. TSA-tree: A wavelet-based approach to improve the efficiency of multi-level surprise and trend queries on time-series data. In *International Conference on Scientific and Statistical Database Management (SSDBM)*, 2000.

[SWJR07] Xiuyao Song, Mingxi Wu, Christopher Jermaine, and Sanjay Ranka. Statistical change detection for multi-dimensional data. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2007.

[Tay03] James W. Taylor. Short-term electricity demand forecasting using double seasonal exponential smoothing. *The Journal of the Operational Research Society*, 2003.

[Tay10] James W. Taylor. Triple seasonal methods for short-term electricity demand forecasting. *European Journal of Operational Research*, 2010.

[TEM07] James W. Taylor and Patrick E. McSharry. Short-term load forecasting methods: An evaluation based on european data. *IEEE Transactions on Power Systems*, 2007.

[TMW04] Kari Torkkola, Noel Massey, and Chip Wood. Driver inattention detection through intelligent analysis of readily available sensors. In *IEEE Conference on Intelligent Transportation Systems*, 2004.

[TO09] Yingying Tao and M Tamer Ozsu. Mining data streams with periodically changing distributions. In *ACM Conference on Information and Knowledge Management*, 2009.

[tpc] Tpc benchmarks. www.tpc.org/information/benchmarks.asp Accessed 1 April 2015.

[TY06] Jun-ichi Takeuchi and Kenji Yamanishi. A unifying framework for detecting outliers and change points from time series. *IEEE Transactions on Knowledge and Data Engineering*, 2006.

[TZ83] Asher Tishler and Israel Zang. A min-max algorithm for non-linear regression models. *Applied Mathematics and Computation*, 1983.

[U.S09] U.S. Department of Energy. The smart grid: an introduction. Electronic Publication, 2009.

[VGT+14] Andreas Veit, Christoph Goebel, Rohit Tidke, Christoph Doblander, and Hans-Arno Jacobsen. Household electricity demand forecasting: Benchmarking state-of-the-art methods. In *International Conference on Future Energy Systems*, 2014.

[VHGK03] Michail Vlachos, Marios Hadjieleftheriou, Dimitrios Gunopulos, and Eamonn Keogh. Indexing multi-dimensional time-series with support for multiple distance measures. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003.

[VVHC+10] Willem Vereecken, Ward Van Heddeghem, Didier Colle, Mario Pickavet, and Piet Demeester. Overall ict footprint and green communication technologies. In *International Symposium on Communications, Control and Signal Processing (ISCCSP)*, 2010.

[VWSK10] Harald Vogt, Holger Weiss, Patrik Spiess, and Achim P. Karduck. Market-based prosumer participation in the smart grid. In *International Conference on Digital Ecosystems and Technologies (DEST)*, 2010.

[Wat15] Watts up? Meters. https://www.wattsupmeters.com, Accessed 22 March 2015.

[WEA13] Tri Kurniawan Wijaya, Julien Eberle, and Karl Aberer. Symbolic representation of smart meter data. In *Workshop on Energy Data Management (EnDM)*, 2013.

[WP90] Ynjiun P. Wang and Theo Pavlidis. Optimal correspondence of string subsequences. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1990.

[YF00] Byoung-Kee Yi and Christos Faloutsos. Fast time sequence indexing for arbitrary L_p Norms. In *International Conference on Very Large Data Bases (VLDB)*, 2000.

[ZL77] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 1977.

[ZSG$^+$03] John Zedlewski, Sumeet Sobti, Nitin Garg, Fengzhou Zheng, Arvind Krishnamurthy, and Randolph Wang. Modeling hard-disk power consumption. In *USENIX Conference on File and Storage Technologies*, 2003.