Analysis of the Parameter Configuration of the Ramer-Douglas-Peucker Algorithm for Time Series Compression

Bachelor Thesis

to attain the scientific degree

Bachelor of Science

handed in by **Linus Heinzl**

Supervisor: Prof. Dr. Tobias Friedrich
Advisor: Philipp Fischbeck
Advisor: Martin Schirneck

Abstract

In this work, we look at the Ramer-Douglas-Peucker algorithm and its use for compression of time series data. We analyze the effect parameter choices have on the algorithm's result. For this, we look at a modified version of the algorithm that can work with multiple ε for a time series. We introduce a heuristic that is able to calculate these multiple ε for a series. For the heuristic, we optimize its parameters with multi-objective optimization, with min-max normalized rMSE and compression factor as objectives. We see that with multiple ε per series, the user can set the heuristic's parameters such that outliers are not kept in the compression. We observe that depending on the dataset, the resulting pareto front displays differences in regards to its shape, which can have linear, concave, or convex characteristics. To approximate the pareto front before compressing a series, we propose to look at the central moving average's similarity to a time series in dependence of the amount of points, over which it is calculated.

Zusammenfassung

In dieser Arbeit beschäftigen wir uns mit dem Ramer-Douglas-Peucker Algorithmus und seiner Verwendung zur Komprimierung von Zeitseriendaten. Wir analysieren die Auswirkung, die Parameter auf das Ergebnis des Algorithmus haben. Dazu betrachten wir eine modifizierte Version des Algorithmus, die auf einer Zeitserie mit mehreren ε arbeiten kann. Wir führen eine Heuristik ein, die diese mehreren ε für eine Serie berechnet. Für die Heuristik optimieren wir ihre Parameter mit einer mehrkriteriellen Optimierung, wobei der min-max normalisierte rMSE und der Kompressionsfaktor als Kriterien dienen. Wir sehen, dass der Benutzer bei mehreren ε pro Serie die Parameter der Heuristik so einstellen kann, dass Ausreißer nicht in der Komprimierung bleiben. Wir stellen fest, dass die resultierende Pareto-Front je nach Datensatz Unterschiede in Bezug auf ihre Form aufweist, die lineare, konkave oder konvexe Eigenschaften haben kann. Um die Paretofront vor dem Komprimieren einer Serie zu approximieren, schlagen wir vor, den Abstand des gleitenden Mittelwertes zu einer Zeitserie in Abhängigkeit von der Anzahl der Punkte zu untersuchen, über die er berechnet wird.

Contents

1 Introduction

With the progress information technology made in the last 30 years, the amount of its possible applications is continuously growing. For a vast amount of fields, we only see the beginning of what is achievable. One of those fields is automation, in which computers perform work that is ordinarily done by humans. For automation, one task is the autonomous monitoring of the machines in a factory to review whether they are functioning properly. If this is not the case, appropriate measures can be taken, without the need of a human to interfere. To monitor the machines, their properties are measured by sensors and then analyzed [16]. Those sensor values could include a machine's temperature, its vibrations or the rotations of its engine. Since these are measured over time, they are called a time series. When there is a large amount of sensor data, there are significant efforts necessary to send and process the data. In order to still deal with the data quickly, more sophisticated and expensive hardware could be bought. There does, however, exist another way to face this situation. The series generated by the sensors can be compressed and therefore reduced in their size [24] [9].

There are two possible ways to compress a time series, lossless compression and lossy compression. With lossless compression, the original data can be completely reconstructed from the compression. Lossless compression consists of techniques such as delta encoding [25] or run length encoding [15]. As a contrast to lossless compression, lossy compression allows that parts of the original data are not reconstructable anymore. When this is the case, there exists the possibility to compress a time series even further. This argument is also made in a survey on lossless and lossy compression methods [31]. To get a handle on a large amount of sensor data, lossy compression therefore becomes handy.

A lossy compression algorithm for a time series is designed such that it gets the series as input and outputs a result that does need significantly less space.

The smaller the size of the compressed series, the easier it is for the data to be sent and processed. However, the loss of information must not be too profound. For instance, when a machine's temperature is measured to control whether it overheats, the compression must not deviate too far from the original temperature values. Otherwise, the monitoring algorithm could falsely state that a machine is overheating, or, even worse, does not recognize when this is indeed the case. When the compression leads to false decisions, a compression algorithm would be rendered useless. But, what data is a compression algorithm allowed to forget? This depends largely on the type of the data, as well as the use case for it. For some analytic tools, a very broad representation of the sensor values may be enough. In other cases, such as the temperature measurements, it may be sensible to keep most of the values and especially sudden peaks. Concluding, there does not exist one single lossy compression setting to solve every problem. Since so much depends on the use case, the user needs to give some input to the compression algorithm that indicates how the algorithm should act.

This input happens in the form of so-called parameters. With parameters, the user can control the behavior of an algorithm. For instance, there exist algorithms where the user can indicate the factor by which the size of the compressed series gets reduced [30] [7]. However, if the parameters require too much a priori knowledge of the user, they may not bring additional utility. For instance, the user may not know to what extent the time series should be compressed, since they do not know the exact appearance of it. They could, however, know the measuring inaccuracy of the sensor that generates the series and therefore the error they are willing to allow. For those use cases, algorithms are developed that have an error bound as parameter [9] [14] [13] [29].

In general, there exist certain challenges when dealing with algorithms that take parameters as input. Coy et al. [10] noticed that for algorithms with parameters, the research community seldom sets these such that they achieve the desired effect, but rather in an "ad hoc way". For the compression algorithms with an error bound as input, the compression may not fulfill its desired use case when the user sets the parameters this way. This holds especially true when they do not know about information such as the sensor's measuring inaccuracies. Also, in the field of data mining, Keogh et al. [20] showed that the performance of a parameter setting for some "parameter-laden" algorithms tends to be entirely different between similar datasets. In regards to compression, it can be vital that this is not the case, particularly when the user wants

to compress multiple time series in one run.

To deal with these challenges, we developed a heuristic in the project that calculates an error tolerance suiting to the series' movements. In this work, we introduce a change to the heuristic such that it can, for one time series, calculate for different areas of it different tolerated errors. Still, the heuristic has parameters that can not be set straightaway. But how exactly does their parameter setting influence the compression? How can the user use the heuristic such that they get a compression that suits their use case? To aid them in this regard, we analyze the heuristic and its parameters in this work.

We show how the heuristic's parameters influence each other and to which value ranges they can be narrowed down. Additionally, we demonstrate how the user can affect the tendency of the compression algorithm to take outliers into account. Furthermore, we show that there are characteristics of the datasets that may highly influence the degree of information loss relevant to the user. To further aid the user in this regard, we propose a simple way to observe those characteristics before a time series gets compressed.

2 Preliminaries

2.1 Introduction to Compression Setup

A compression algorithm gets a time series as input and outputs a compressed version of the time series. In order to work with the algorithm, we define the term *time series* first. We define a time series as a sequence of points. Each point consists of two numbers, its time $t \in \mathbb{N}$, and its value $y \in \mathbb{R}$. The sequence is ordered by the time of the points. Let T be a time series. With T_i we denote its i-th point. Furthermore, we define $y(T_i)$ as the value of the i-th point. With $t(T_i)$, we denote the time of the i-th point. We denote the amount of points that are contained in T with $|T|$.

To compress a time series, we remove points from it. We can therefore use the notation defined above as well for a compressed series. For it, we demand that each of its times must also exist in the original time series. Let T be a time series and let C be its compressed series. To fulfill this requirement, we demand that $\{t(C_i) \mid i \leq |C|\}$ is a subset of $\{t(T_i) \mid i \leq |T|\}$. However, to allow for more flexible compression algorithms, we allow that the value y of a point gets changed. Therefore, $\{y(C_i) \mid i \leq |C|\}$ is not necessarily a subset of $\{y(T_i) \mid i \leq |T|\}$. We define the compression factor of C as $\frac{|T|}{|C|}$. The less points are contained in C, the higher is its compression factor.

To compare the compressed version of a time series with the original version, we need to measure its similarity to the original series. In order to do this, we compare the time series to a reconstructed version of the compressed time series. Let T be a time series, C be its compressed version, and R be its reconstructed version. Then the reconstructed time series has the following properties:

- Both time series, the reconstructed and the original, have the same length. Therefore, $|T| = |R|$
- The reconstructed series consists of the same times as the original series. It holds that $\forall i \leq |T| : t(T_i) = t(R_i)$
- The y-values of the reconstructed are obtained by the use of linear interpolation. Let R_i be a point of the reconstructed time series R. We know that the time of R_i is equal to $t(T_i)$. To calculate its value y, we use linear interpolation between two adjacent points of C. We call those two points C_a and C_{a+1}. They are chosen, such that the time of

R_i is located between the times of them. More precisely it holds that $t(C_a) \leq t(R_i) \leq t(C_{a+1})$. The value $y(R_i)$ is now calculated as

$$y(C_a) + (t(R_i) - t(C_a)) \cdot \frac{y(C_{a+1}) - y(C_a)}{t(C_{a+1}) - t(C_a)}$$

.

2.2 Compression Algorithm

In the bachelor's project we researched existing compression algorithms, developed new algorithms, and compared which algorithm was most suitable to our needs. In order to do this, we had several criteria in order to evaluate the algorithms. First of all, the algorithm should keep points that represent the series fairly well. Secondly, for large amounts of data, we needed the algorithms to work in a reasonable time. Regarding those criteria, the Ramer-Douglas-Peucker algorithm performed very well in the project, which is why we use it here in order to conduct our experiments.

Ramer-Douglas-Peucker Algorithm

The Ramer-Douglas-Peucker algorithm [12] is mainly used in the context of line simplification. In line simplification, an algorithm gets a polygon as input and outputs a polygon that has fewer points than the original polygon, while being as similar to it as possible. With slight modifications, we used this algorithm in the project. In our setting in the project, the algorithm gets a time series and an $\varepsilon \in \mathbb{R}^+$ as input. It outputs a compressed time series.

With the parameter ε, the user can specify the maximum deviation that they want the algorithm's result to allow. Let T be a time series and let R be the reconstructed time series of the compression algorithm's result. For an $i < |T|$, the deviation to the i-th point in T is defined as $|y(R_i) - y(T_i)|$. The maximum deviation is defined as $\max(\{|y(R_i) - y(T_i)| \mid i \leq |T|\})$. For an ε, the algorithm compresses the series such that the maximum deviation to T is smaller or equal than ε. We therefore say that with ε, the user can set the allowed maximum deviation of an algorithm. Similarly, we define the maximum deviation of a point to a line. Let T be a time series, let l be a line, and let $l(x)$ be the y-value of the line at x. Then, the deviation of the i-th point T_i to the line is defined as $|y(T_i) - l(t(T_i))|$. $l(t(T_i))$ is equal to the line's value at the i-th point's time.

Ramer-Douglas-Peucker is a divide-and-conquer algorithm. In its initial call, it gets an ε and the time series as input. Then, it draws a line between the first and last point of it. Afterwards, it searches for the point in the time series that has the maximum deviation to the drawn line. We denote this point as T_m. If this maximum deviation is smaller than ε, it returns a compressed series that only contains the first and last point. Otherwise, it divides the time series into two new parts. The first part starts at the first point and contains every point until T_m. Similarly, the second part contains all points from T_m until the last point. For each of those two parts, the algorithm calls itself recursively, with the part and ε as input. In the next step, it merges the compressed series that were returned from the two recursive calls and returns them.

Modifications of Ramer-Douglas-Peucker Algorithm

In the established Ramer-Douglas-Peucker algorithm, the user specifies one ε as parameter. The algorithm compresses the series such that the deviation to every point in T is smaller than ε. However, it can be sensible to allow a higher deviation to some points in T and a smaller deviation to others. For example, there can be areas in the series, which have a high amount of fluctuations. Depending on the use case, it is possible that the user would not want to keep these oscilliations in the compressed time series. However, if they select ε accordingly, they could lose information interesting to them in other parts of the series.

In order to handle these situations, we slightly altered the Ramer-Douglas-Peucker algorithm in the bachelor's project. In the altered version, the algorithm gets a time series T and an array E of size $|T|$ as input. For each i, $E[i]$ contains the allowed deviation to the point T_i. In the previous setting, the algorithm calculated the maximum deviation to the line to decide whether the time series needs to be compressed. However, in this altered setting this does not work. Even, when for a point T_i the deviation to the line is smaller than its allowed deviation $E[i]$, there could exist other points in the series that have a higher deviation than they are allowed to.

To address this problem, we calculated the maximum relative deviation instead of the maximum deviation in the project. To calculate a relative deviation of a point T_i to a line l, we compute

$$\frac{|y(T_i) - l(t(T_i))|}{E[i]}$$

If the maximum relative deviation is below one, only the first and last point of T are contained in the compressed series returned by the algorithm. Otherwise, we split the time series by the point with the maximum relative deviation and call the algorithm recursively, similarly to the original algorithm.

Local Optimization

In the bachelor's project, we developed an algorithm that further optimizes a compression's result. This algorithm is called *LoOp*, which is an abbreviation for local optimization. Let T be an original time series and let C be a compressed series that is output by an algorithm that compresses T. LoOp moves the points in C such that the rMSE (see Section 2.5) of the reconstructed series from C gets improved. For this, LoOp iteratively considers three neighboring points of C. It starts with the first three points, then it looks from the second to the fourth point, and so on. At each step, LoOp attempts to improve on the rMSE. To do this, it fixes the first and last point and changes the middle point's time and value, such that the resulting rMSE of the reconstructed series is minimal. The mathematical fundamentals of this algorithm are further discussed in [32]. Since no new points are added, $|C|$ stays the same after the application of LoOp. Also, LoOp moves the points such that their times are a part of the original time series as well.

A compressed time series can contain points whose placement leads to a high error. When the error can be considerably reduced by slightly altering these points, LoOp may be able to do this. Therefore, LoOp is able to significantly improve on the rMSE in some cases. However, since LoOp changes the points in C, we can not guarantee that the deviation to every point in T is smaller than allowed. Nevertheless, since LoOp optimizes for the rMSE, we expect that the deviation to the vast majority of points is still within the tolerance. Otherwise, the rMSE would be increased, which is not possible per LoOp's definition. For the experiments, we always use the version of the Ramer-Douglas-Peucker algorithm that can handle multiple ε. Afterwards, we apply LoOp as a way to make sure the compressed series does not contain points with values that are very unfavorably placed in regards to the rMSE.

2.3 Determing Parameter ε

As we discussed, the Ramer-Douglas-Peucker algorithm gets an ε as parameter. It is important that the user chooses an ε suitable to their needs. The higher

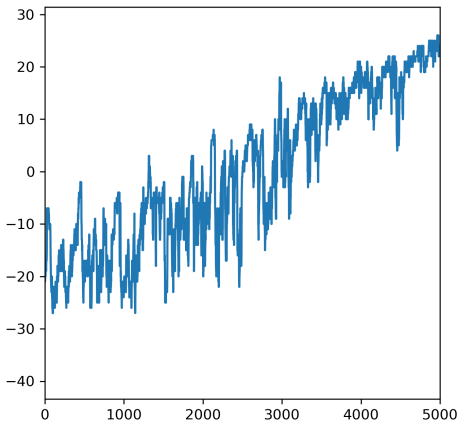
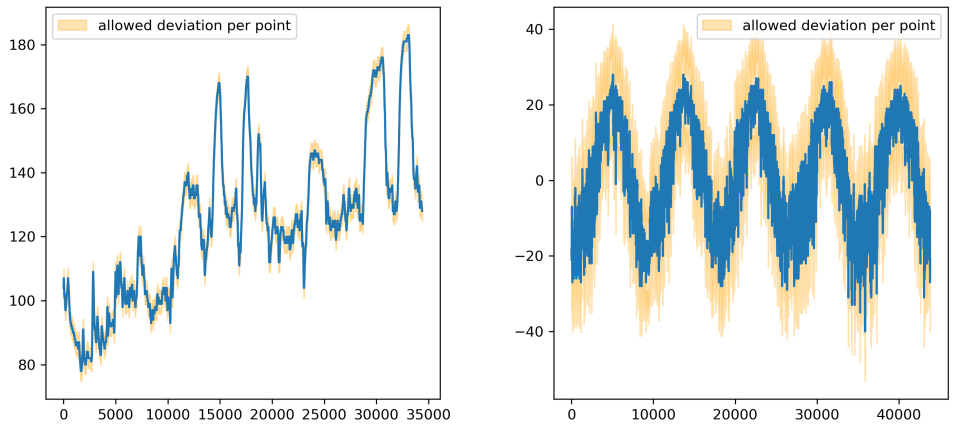


Figure 2.1: This Figure shows a window of a time series of the Beijing $PM_{2.5}$ dataset [22], consisting of 5000 points. In this area, there is an upwards trend discernible, but also many fluctuations.

ε, the less accuracy the compressed time series has. Therefore, an ε not well chosen leads to unsatisfactory results to the user. For one time series, a user with expert knowledge might know what deviation ε they want to tolerate, for instance because they know the measuring inaccuracy of the sensor. However, there might be series where this is not the case. Especially when there are similar series that should be compressed, a user might not want to choose an individual ε for each time series. However, to set the same ε for all series may also not be an option, since those time series can have slightly different characteristics and varying value ranges. In our bachelor's project, we solved this problem by calculating ε with a heuristic for each series.

In time series analysis, time series are usually decomposed into multiple components [4]. In the decomposition, they are divided into trend, seasonal patterns, cyclic patterns, and an irregular component. A trend shows the direction of a time series over a long period of time. Cyclic patterns are behavior of a time series that repeat themselves, but not with a fixed period. On the other hand, seasonal patterns occur over a fixed period. Irregular movements are movements of the series that can not be explained with the other three components. We denote these movements, which often show a lot of fluctuations and variations, as noise (see Figure 2.1). For the heuristic, we assume that the user usually would like to keep movements such as the trend, but is not interested in noise.



(a) (b)

Figure 2.2: In Figure 2.2a, a time series of the Pamap2 dataset [27] is displayed. Figure 2.2b shows a time series of the Beijing $PM_{2.5}$ dataset. Both series have a similar amount of points. For 2.2a, the heuristic calculates a considerably smaller ε.

To explain the heuristic, we define the term *window* first. A window is a subset of the original time series, in which consecutive points of the series are contained. We say a window has size w when it consists of w points. We can partition a time series T into windows. For a partition, each point in T needs to be contained in exactly one window. When every window of the partition except the last one has size w, we denote that we partitioned the series into windows of size w.

To calculate ε, we partition the series into windows of size w. The parameter w is chosen by the user. It indicates that to them, each window of size w mostly consist of noise. With a large window size, the user indicates that most of the series consists of movements they consider noise. On the other hand, a small window size suggests most of the series' movements are important to them. With this information in mind, the heuristic calculates the parameter ε for a time series.

Per window, it calculates a maximum deviation d of the noise to the window's regular movements. We tried out multiple approaches to approximate this, which are discussed in the next section. Then, we aggregate every calculated

d by taking the median of them. For the time series, this calculated median is then our result for its allowed deviation ε. When the user wants to compress multiple time series, they may not need to set every ε by hand anymore. With the heuristic, they can set the parameter w once, and then get approximated ε for each time series. The heuristic sensibly adapts to certain characteristics of the time series. For instance, when a series has a higher amount of noise, it calculates a higher ε (see Figure 2.2).

Calculating multiple ε

In the project, the heuristic calculated one allowed maximum deviation ε for each time series. However, in a time series with different noise levels, it could be sensible to allow for multiple ε per series. In the project, we took the median over the calculated divergences d of the noise to the trend, and set ε to the result. In this work, we change this such that the allowed deviations ε can differ between each window. Per window, we set an allowed deviation ε that only applies to its own points. Naturally, the divergence d of the window's noise to the trend influences its own ε. Additionally, to allow for flexibility, the user can still aggregate the values of d from multiple windows, in order to calculate an allowed deviation ε for the window's points. With the parameter s, they can specify how many windows are taken into account for that. Set to the smallest value, the user indicates that the allowed deviation ε for each window should be equal to the value d calculated for it. When they set s to the largest value, they state that the calculated divergences should be aggregated over every window to calculate an ε, just as in the heuristic used in the project. The parameter s has a relative value, with a value range between 0 and 1. Let w be a window size, and P be a partition of the series into windows of size w. Let W be a window on which we want to calculate for its points the allowed deviation ε to them. Then, to each side of W, the values of d from its nearest $s \cdot |P|$ neighboring windows are taken into account and then aggregated. Since both sides of the window, the left and right, are taken into account, the ε for W is calculated from $2 \cdot s \cdot |P|$ values, and additionally from the value d of the window itself. These values are aggregated by taking the median over them.

Also, there can be cases where the user wants to slightly increase the intensity of the compression, while still retaining the current partition of the windows. In the project, we therefore introduced that they can choose one constant factor f, by which each calculated ε gets multiplied.

For the experiments that we conduct, we use this altered heuristic. Therefore,

there are three parameters that we analyze, the window size w, the relative size s of the set of neighboring windows from which an ε is calculated, and the scaling factor f.

2.4 Window Heuristics

Let w be a window size and let T be a time series. Then, we denote with W a window of T that has size w. Given such a window W, we need to calculate the maximum deviation d of its noise to its regular movements. For this, we have tried out multiple approaches in the project.

Min-Max

The *min-max heuristic* is quite straightforward. With $P = \{y(W_i) \mid i \leq w\}$, we denote the values of the window's points. The value range of W is then defined as $\max(P) - \min(P)$. With the min-max heuristic, we set d to the value range of the window. We know that the maximum deviation to the regular movements is always smaller than the value range of W. Additionally, when there exist more oscillating movements in the window, this is often reflected by a higher value range, which leads to a higher calculated ε. Therefore, this approach often allows for a higher compression factor, when there are more fluctuating movements in a window. However, the heuristic can in some instances lead to unwanted behavior. For instance, we can consider a window where the values of the time series only only consists of a clear trend, and not of any noise. Here, we would want the heuristic to calculate a small deviation d. This is not the case here. Since only the value range gets considered, the heuristic would compute a large deviation d in those cases.

Linear Regression

To better deal with steep slopes, we can compute the deviation d with other heuristics. One heuristic, the *linear regression heuristic*, approximates the trend line in the window (see Figure 2.3). The trend line is calculated by simple linear regression over all points in W. This regression line has the form $m \cdot x + b$. We denote the first point of the window with W_1, and the last point with W_w. $y(W_i)$ indicates the value of a point, $t(W_i)$ its time. $\bar{t}$ denotes the mean of the window's times, $\overline{y}$ is equal to the mean of the y-values in the window. We apply simple linear regression as described in [3]. For this, the

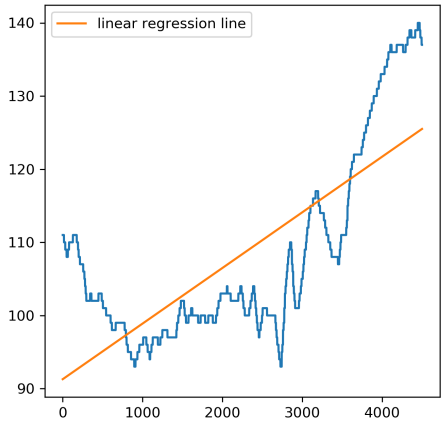


Figure 2.3: A time series of the Pamap2 dataset[27] is shown. Given a window of size 5000, the trend of the series is approximated by linear regression.

slope m of the linear regression line is set as

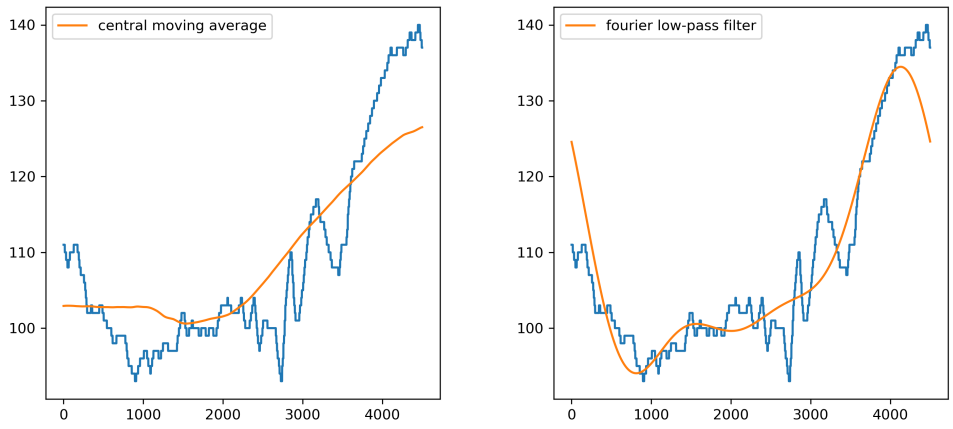
$$\frac{\sum_{j=1}^{n}(t(W_j)-\overline{t})(y(W_j)-\overline{y})}{\sum_{j=1}^{n}(t(W_j)-\overline{t})^2}$$

We set its constant value b as $\overline{y}-a\cdot\overline{t}$.

For the window, we compute per point the deviation to the trend line. The maximum deviation to the trend line is seen as the deviation of the irregular movements to the trend. Applied on a window with a high slope, the calculated deviation d is therefore small as well.

Central Simple Moving Average

For a window, we want to separate its trend from its irregular movements. Another established way to filter these is to calculate a moving average on the series [21], see Figure 2.4. For each point W_i of the window, we calculate its averaged value as the mean of $\{y(W_a) \mid |a-i| < \frac{2}{3}\cdot w\}$. The series that consists of those averaged values is called *central simple moving average* (CSMA) [18]. $\frac{2}{3}$ was determined empirically on the public datasets that are introduced later in this work. With this value the CSMA follows the movements in the window, but it does not follow every small movement in it. We define for a point W_i its deviation to the CSMA as the absolute difference between the point's value



(a) (b)

Figure 2.4: The same window as in Figure 2.3 is shown. In 2.4a the trend is approximated with the CSMA. In 2.4b, it is approximated with a low-pass filter using Fourier transformation.

and its averaged value. To compute d, we calculate for each point W_i this deviation. Then, we set d as the maximum of the computed values.

Fourier Transform

In addition to the CSMA, there are other approaches to filter oscillating movements that are mentioned in literature [21]. One idea that is discussed, is to apply *Fourier transform* on the data. Every series can be constructed from a sum of sine waves. By applying Fourier transform, we transform the time series into the frequency domain. The mathematical foundations for that are described in [17]. The frequency domain displays to a given frequency, how much the sine wave with this frequency contributes to the original signal. We denote this as the frequency's *amplitude*. The higher the frequency, the more volatile the accompanying sine wave is. Since we want to have a smoothed time series, we do not want that waves with a high frequency contribute to the signal. In the frequency domain, we can filter the data, as is described in detail in [1]. To filter the data, the series in the frequency domain is changed, such that the amplitudes of frequencies above a certain *cut-off frequency* are set to 0. Since the frequencies are discrete values, we can specify how many of them should be kept at their original amplitude. We denote this amount as c. In the calculation, we keep the amplitudes of the c lowest frequencies at their original value, while they are set to 0 for the rest of the frequencies. To only keep the

sine waves with the lowest frequencies, we set c to a small number, with $c = 5$. Similarly to the CSMA heuristic, this number was determined empirically with the goal that only the broadest movements of the series are displayed in the smoothed time series. The Fourier transform has the characteristic that it can be inversed [17]. With the inversion, we are able to transform a series from the frequency domain into the time domain, again. After we set the amplitudes of the vast amount of frequencies to 0, and apply the inverse Fourier transform, we have a smoothed time series with less oscillations (see Figure 2.4). This procedure is called *low-pass filter*. On the smoothed time series, we can now measure the maximum deviation between the smoothed series and the original series and set it as our calculated deviation d for the window. The maximum deviation to this smoothed series is calculated analogously to the deviation to the CSMA.

2.5 Error Measures

When we compress a time series, we need to measure the quality of the compression. For this purpose, there exist many error measures, some of them are for example discussed in [28] [5]. For the experiments, we use some of the measures discussed in these works. For each of the error measures, we calculate the error of a reconstructed time series R to a time series T.

Mean Absolute Error

A simple error measurement is the *mean absolute error* (MAE). The MAE aggregates the deviation of the points. The smaller the aggregated deviation is, the more similar the reconstructed time series is to the original time series. The MAE is equal to

$$\frac{1}{n} \cdot \sum_{i=1}^{|T|} |y(T_i) - y(R_i)|$$

Root Mean Squared Error

To calculate the *root mean squared error* (rMSE), we calculate the *mean squared error* (MSE) first. MSE is defined as

$$\frac{1}{n} \cdot \sum_{i=1}^{|T|} (T_i - R_i)^2$$

Since each point's deviation is squared, MSE further punishes it when the reconstructed time series contains points with a high deviation. On the other hand, small deviations receive less importance. To compute the rMSE, we take the root of the MSE. This way, the error has the same dimension as the MAE.

L_∞ Error

The L_∞ error only considers the point with the highest deviation. It is calculated as

$$\max(\{|T_i - R_i| \mid i \leq n \})$$

As a consequence, every point except the point with the largest deviation has no influence on the L_∞ error.

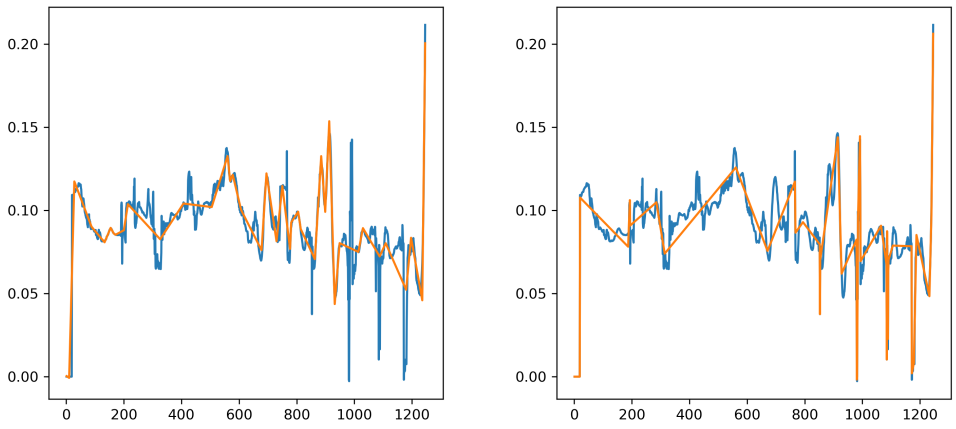
2.6 Normalization

We need to consider that the time series can have different value ranges. For instance, there exist series with a range from 0 to 1. Other series may have a range from 0 to 2000. To compress a time series with a high value range may result in a greater error than when a series with a small range is compressed. Especially when we compress multiple time series, this property would make it hard to compare errors of the reconstructed series to each other. To reduce the dependency on the series' value range, we normalize each of the calculated errors by it. Let v be the value range of the series, and e an error that is calculated on the reconstructed series. Then, the normalized error is calculated as $e \cdot \frac{1}{v}$. We call this normalization the *min-max normalization*. We denote the normalized error measures with nMAE, nRMSE, and nL_∞, respectively. The error can have values between 0 and 1. In the experiments, we saw for each error measure that an error smaller than 0.01 can be considered as small. On the other hands, except for the nL_∞ norm, the normalized errors seldom displayed a value above 0.15. For the nL_∞ norm, the normalized errors could be as large as 0.5.

There exist other ways to define error measures that do not measure absolute deviations. For example, there exists the category of relative error measures. In a relative error measure, the error to a point T_i is measured in relation to the point's value. When we deviate to a point with a large value, the error is smaller, compared to when we have the same deviation to a point with a small value. This property, however, is not attractive for our compression setting. On the one side, compression on a time series with a value range between 1000 and 1010 leads to low errors. On the other side, if the values of a time series range between 0 and 10, a similar compression can result in an error that is quite high. Therefore, the relative error for a compression can vary a lot, depending on the value range of the series.

Comparison between the normalized Error Measures

In order to decide how to measure the error, we discuss the differences between these normalized error measures. The main difference between nMAE, nRMSE and nL_∞ is their punishment of outliers. We view outliers as points in a time series, whose value differs greatly from the value of the points in their near environment. Since the nRMSE penalizes high deviations of the reconstructed series to the original series' points, a compression algorithm often needs to take outliers into account to achieve a small error. In contrast to this, ignoring



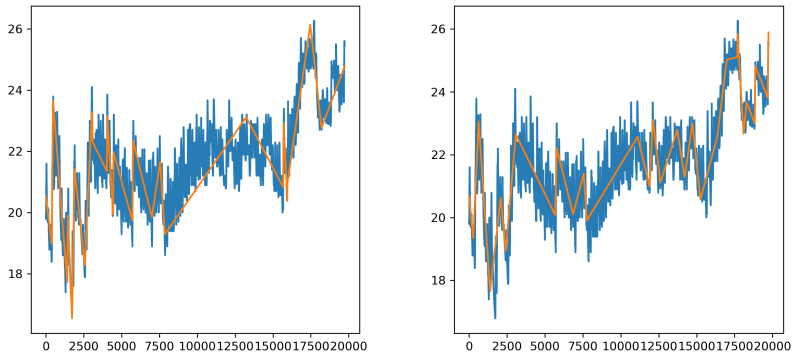
(a) (b)

Figure 2.5: Two settings to compress a time series of the Kinect dataset [2]. In the first setting, ε is different per window, while in the second setting one ε is calculated for the whole series. The reconstructed series of the first setting is shown in Figure 2.5a, while Figure 2.5b displays the result of the second setting. In the second setting, outliers are kept, while the first setting keeps other movements instead of outliers. The second setting leads to a lower nRMSE of the reconstructed series (0.04 instead of 0.05), while the first yields a lower nMAE (0.028 instead of 0.031).

outliers does not necessarily lead to a high nMAE. This can be seen for instance, when we compare two different compression settings (see Figure 2.5). The first setting keeps outliers, while the other keeps all parts of the time series that have a persistent slope. Both instances have the same compression factor. Nevertheless, the reconstructed series of the first compression has a higher nMAE than the one of the second, while in regards to the nRMSE, it is the other way around.

Thus, the choice of a suitable error measure depends on the user's preference. Do they want to keep all outliers? Then, it may not be a good idea to measure the error with nMAE. Are they not interested in them? Then, using nMAE as error measure may become a good idea. For the experiments, we mainly look at the results that we get with the error measure nRMSE, however, we bring up the nMAE as comparison when suitable.

In contrast to nMAE and nRMSE, nL_∞ only depends on one point of the reconstructed time series, namely the point that has maximal deviation. Therefore,



(a) (b)

Figure 2.6: Two settings to compress a time series of the Energy Production dataset [8]. In the first setting, one ε is calculated for the whole series, while in the second one ε is different per window. The reconstructed series of the first setting is shown in Figure 2.5a, while Figure 2.5b displays the result of the second setting. The first setting leads to a lower nL_∞ error of the reconstructed series (0.28 instead of 0.29), while it has a clearly larger nRMSE (0.082 instead of 0.072).

to have a small error, it is sufficient when there is a small maximum deviation. However, when the maximum deviation is not large, this does not necessarily mean that most of the points have a small deviation (see Figure 2.6). If the user wants the properties of the entire reconstructed series to be covered by the error measure, $\text{n}L_\infty$ is not suitable. In our experiments, we only bring up the nL_∞ error when we compare how the error measure influences the non-dominated parameters.

3 Experimental Setup

3.1 Multi-Objective Optimization

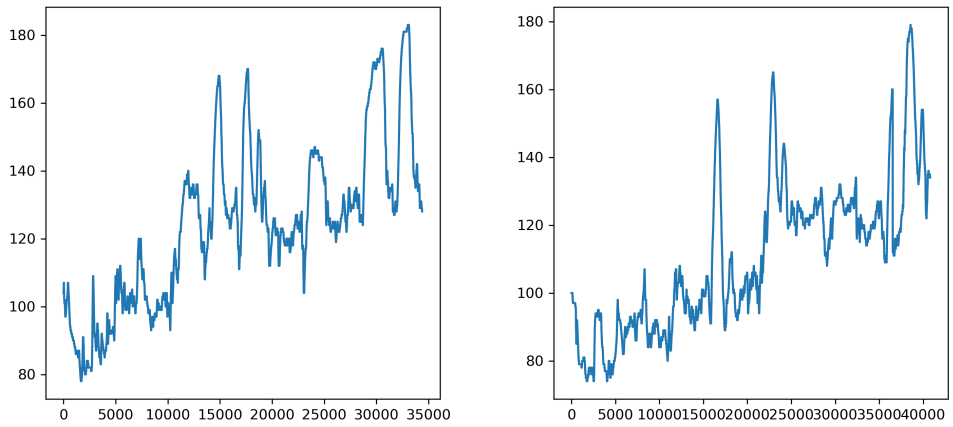
We analyze the effect of the different heuristics and parameter choices on the results of the compression algorithm. To evaluate a fix setting of heuristic and parameters, we introduce datasets to run it on. This happens in Section 3.2. Any given parameter setting is applied to every series in the respective set. This way, the results are more generalizable than if we would have ran the algorithm on a single time series only.

To evaluate the result of a fix setting, we define quality measures. When we analyze a setting, there are two criteria that we consider. The first criterium is the aggregated error we get when the series in the dataset are compressed with the setting. For this, we calculate the errors of the compression's reconstructed series with one of our normalized error functions presented in 2.5 and take the mean over them. When we use, for instance, the nRMSE as error function, this criterium is denoted as *aggregated nRMSE*. The second criterium is the *aggregated compression factor*. For a dataset and fix setting, we take the harmonic mean [26] of the compression factors achieved on the dataset's series. Usually, a larger aggregated compression factor leads to a larger aggregated error. Therefore, there is no compression setting that excels in both criteria.

To still compare them to each other, there exists the technique of multi-objective optimization [19]. With multi-objective optimization, both criteria are taken into account when we optimize the parameters' values. If for a parameter combination, there exists another combination that performs better on both criteria, the parameter setting is *dominated*. Consequently, we can disregard it. Finally, we analyze the parameter combinations that are not dominated.

There exist approaches in literature to find non-dominated parameters, as are for example discussed in [6]. We use the *non dominated sorting genetic algorithm* (NSGA2)[11]. It generates a few random parameter settings in the beginning and calculates which settings produce results that are not dominated. For those settings, the parameter values are then modified and again evaluated afterwards. If these changes produce valuable results again, the procedure is repeated, and so on. In contrast to this, settings that produce a high aggregated error and low aggregated compression factor are disregarded and not used for further calculations. The combinations that are not dominated consist of an aggregated error and an aggregated compression factor. Therefore, we are able to plot them against each other. The resulting graph is called a pareto front.

The experiments were run on a machine with an Intel(R) Xeon(R) Gold 5118 CPU with 2.30 GHz that supports SSE, SSE2, SSSE3, SSE4.1, SSE4.2, and AVX instruction sets. The machine has 62 GB RAM available. It runs Linux 4.14.85-rancher. To generate a pareto front for one dataset took a few hours of time at maximum. In order for the NSGA2 algorithm to generate sensible parameter combinations, we need to limit the ranges of the values that the parameters can take. For the window size w, we set the minimum window size to 10, since with windows for a smaller size than that, it is not feasible to discern irregular movements and trends. For the scaling factor f, we analyze the case that the user wants to increase the degree of the compression without increasing the window size, since we presume this use case may be relevant to them. Therefore we set the minimum value of f to 1. We set the maximum values for w and f to 10,000, respectively 50.0. For each of these values, a very high aggregated compression factor is achieved on the datasets, even independently of the values for the other parameters. Since s is a relative value, we limit its range to be between 0 and 1.



(a) (b)

Figure 3.1: Two example time series of the Pamap2 dataset.

3.2 Datasets

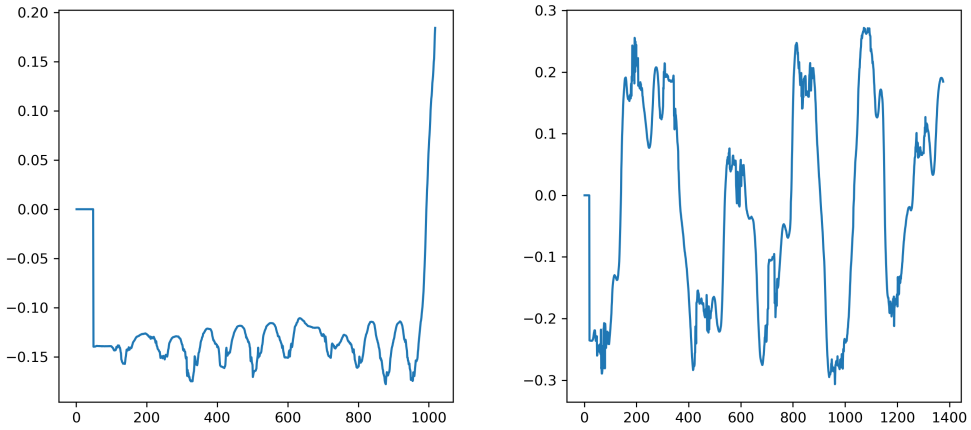
For the experiments, we look at different types of time series. For this, we consider the following publicly available datasets. In section 2.3 we described that a time series can be decomposed into trend, seasonal patterns, cyclical patterns, and an irregular component. Both seasonal and cyclical patterns display behavior that repeats over time, however in cyclical pattern this repetition does not occur over a fixed period. We use this decomposition to analyze the series in the datasets.

Pamap2

For the *Pamap2*[1] dataset, participants were tracked for 10 hours, while they performed physical activities [27]. During this time, their heart rate was observed. Furthermore, their movement was measured at a frequency of 100Hz with inertial measuring units at their hands, chest and ankles. In the experiments, we only look at the nine time series showing the participants' heart rate, since the other series consist mostly of variations and irregular movements.

The series generated by the heart rates contain between 20,000 and 40,000 points, depending on the participant. Example series can be seen in Figure

1 https://archive.ics.uci.edu/ml/machine-learning-databases/00231/



(a) (b)

Figure 3.2: Two example time series of the Kinect dataset.

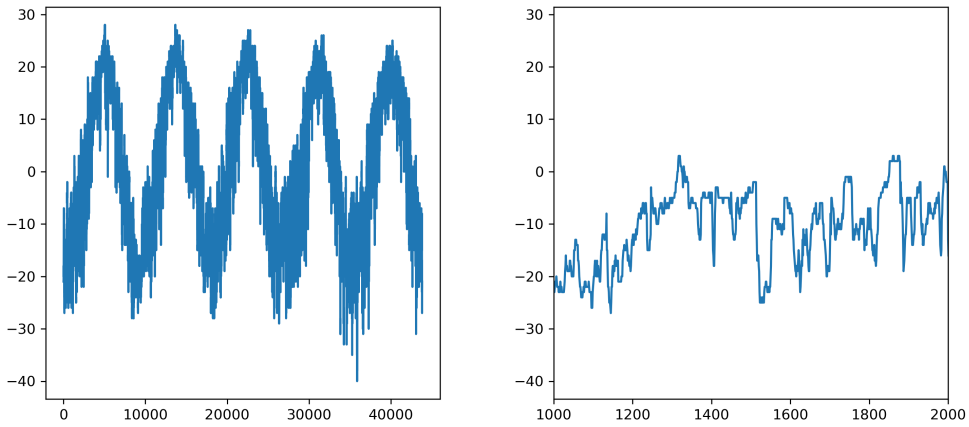
3.1. Typically for the series, a long-term upward trend is discernible. There are no cyclic or seasonal patterns in the series. There are fluctuations in the series, but they only extend to larger intervalls.

Kinect

For the *Kinect*[2] dataset provided by Microsoft, participants were asked to perform several gestures (e.g. "Throw an object", "Change weapon")[2]. While they performed the gestures, the coordinates of several body parts were tracked. Each coordinate forms a time series that can be analyzed and compressed. To ensure some similarity between the series, we only consider the series displaying the values for one coordinate of one body part. For that, we choose the x-coordinate of the participant's spine, however a different coordinate could have been fitting just as well.

Each time series has between 1000 and 2000 points. In the datasets there are 594 gestures performed. Per gesture, we consider one time series (the x-coordinates of the participant's spine). In contrast to the Pamap2 dataset, time series in the Kinect dataset do not show a trend. Furthermore, most of the series in the Kinect dataset have seasonal characteristics (see Figure 3.2 a, 3.2 b). These seasonal patterns occur over a longer timespan. In addition,

2 https://www.microsoft.com/en-us/download/details.aspx?id=52283



(a) (b)

Figure 3.3: Figure 3.3a displays a time series of the Beijing $PM_{2.5}$ dataset in full length. In Figure 3.3b a window of the series of size 1000 is shown.

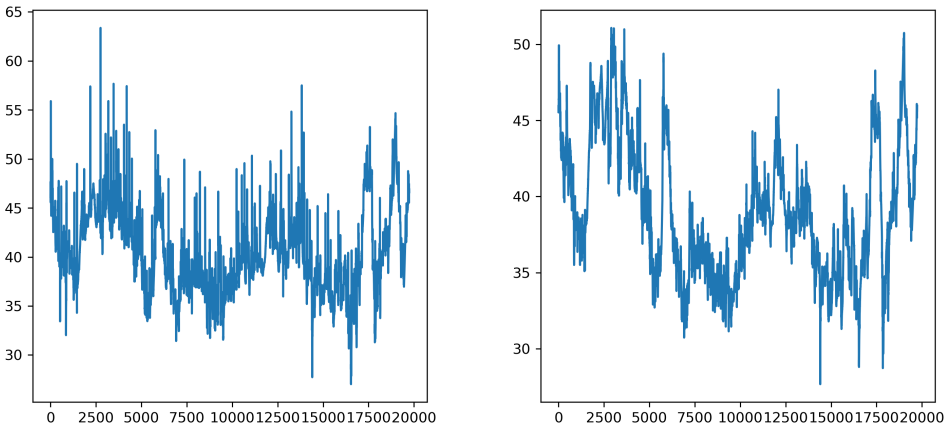
some time series have large peaks at their beginning or end.

Beijing $PM_{2.5}$

In the *Beijing $PM_{2.5}$*[3] dataset, Scientists from the Peking University analyzed the air pollution of Beijing, People's Republic of China. For this, they determined the concentration of the fine particulate $PM_{2.5}$. To generate context information, they also assessed the dew point, temperature and precipitation. The measurements were taken every hour from the beginning of 2010 to the end of 2014. From this set, we investigate the three series generated by the dew point, temperature and precipitation data [22].

These series consist of 43,824 points. Similarly to the Kinect dataset, no trend is discernible in the time series. However, here the series show strong seasonality. In the series (see Figure 3.3a), we see a yearly pattern (one year corresponds to 8760 points). Besides the seasonality, the series are further influenced by irregularities and random movements. Even in a window that only encompasses a small fraction of the series' length, there is a considerable amount of fluctuations discernible. This is shown in Figure 3.3b.

3 https://archive.ics.uci.edu/ml/machine-learning-databases/00381/



(a) (b)

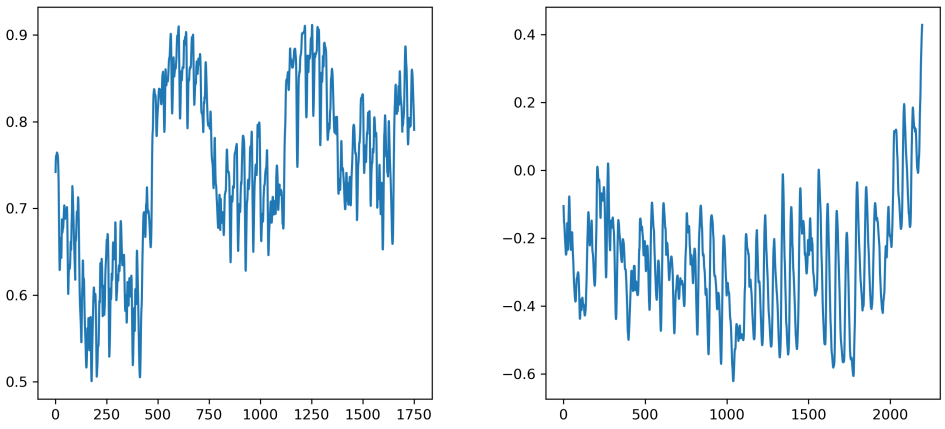
Figure 3.4: Two example time series of the Energy Production dataset.

Energy Production

For the *Energy Production*[4] dataset, scientists of the University of Mons, Belgium, tested prediction models for the energy use of appliances in a household. For this, they measured their energy use in the span of four months, with one reading every ten minutes. As context information, they also took readings of the temperature and humility in each of the household's room. We look at the time series generated by the context information [8].

Each time series consists of 19,735 points. Some of the series in the dataset are exclusively composed of fluctuations and are disregarded. We consider the residual 19 time series. In the majority of them, no major trend is discernible. Furthermore, in similarity to the Pamap2 dataset, no time series contains seasonal or cyclical patterns. There is a large number of fluctuations in the series, even when we only look at a small window (see Figure 3.4). In that regard, the Energy Production series are similar to the Beijing $PM_{2.5}$ dataset. In addition, a lot of the series contain outliers. This phenomenon is more prevalent in this dataset than it is in the other datasets.

4 https://archive.ics.uci.edu/ml/machine-learning-databases/00374/



(a) (b)

Figure 3.5: Two example time series of the Human Activity Recognition dataset.

Human Activity Recognition

For the *Human Activity Recognition*[5] dataset, scientists of the Queen Mary University of London, UK, collected gravity, accelerometer, and gyroscope data of people that executed activities such as walking, jogging or standing [23]. We select the time series such that they display similar patterns. This is, for instance, the case for some series generated by the gravity sensor's y coordinate. In total, we choose eleven series showing this sensor's value. The length of the series varies from 1,000 to 3,000.

Every time series contains seasonal patterns that repeat over a very short period (see Figure 3.5). In some series, there are even cyclical patterns over longer timespans (see Figure 3.5a).

5 https://www.kaggle.com/malekzadeh/motionsense-dataset

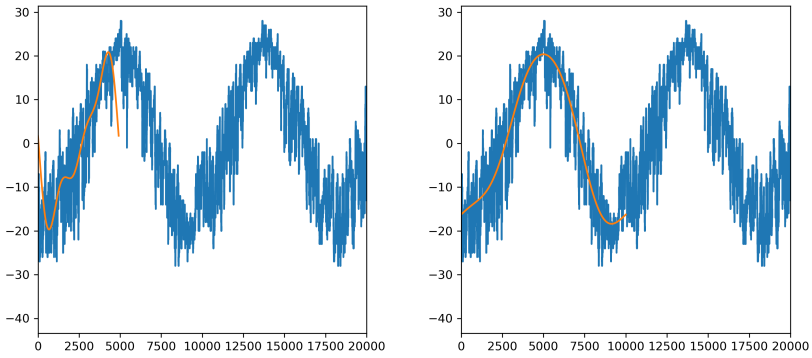
4 Results

4.1 Differences Between the Heuristics

We do not discuss the results for each combination of window heuristic and a dataset. Instead, we discuss the influences of the heuristics first and see whether there are even noticeable differences between them. Are there heuristics that perform better than others in general? Are there interesting patterns? The pareto front displays for a dataset the aggregated error in dependence of the aggregated compression factor, and can therefore be used to analyze a heuristic. Besides the pareto front, there are other things that can be considered when we discuss a window heuristic, such as its robustness.

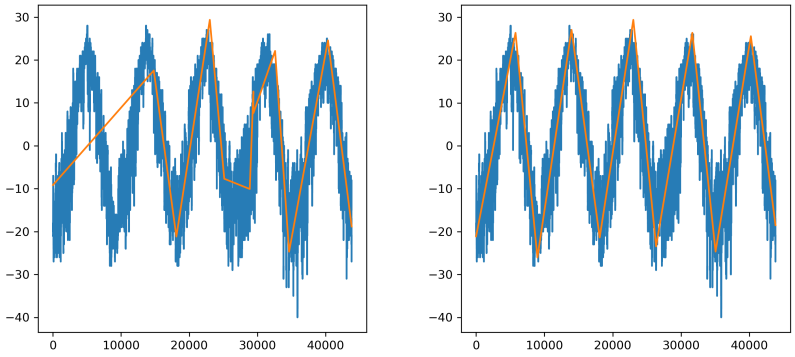
Robustness

With robustness we denote behavior of a compression algorithm when there are slight changes in the parameters. When the result of an algorithm only changes slightly when there are small modifications in the parameters, the algorithm is robust. Furthermore, in regards to robustness, the values of the metrics should go in the intended direction. For example, the window size w influences the computed deviation ε per window. The larger the user sets w, the more movement there is per window, which should reflect in a higher ε. Therefore, the user expects a higher compression factor when they increase the window size. Now, we discuss to what extent the algorithm's behavior meets this expectation when different heuristics are used.



(a) (b)

Figure 4.1: A comparison of the low-pass filtered series that are calculated by the Fourier transform heuristic. In Figure 4.1a, the filtered series for a window of size 4910 is shown. In Figure 4.1b, the low-pass filtered series for a window of size 10,000 is displayed. Despite the significantly smaller window size, the low-pass filtered series in Figure 4.1a has a larger deviation to the original series (23.6 instead of 20.6).



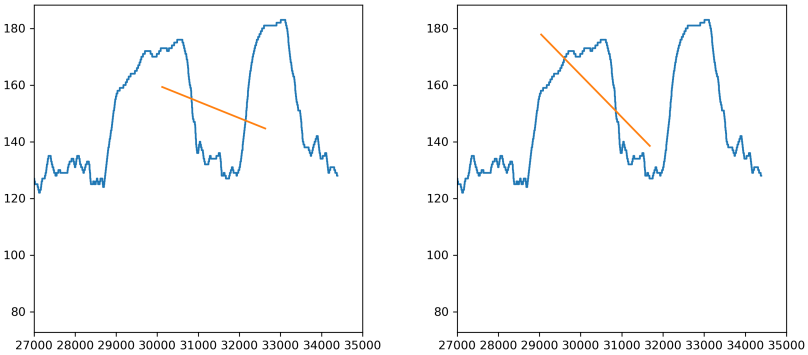
(a) (b)

Figure 4.2: A comparison of two compression settings for the series of Figure 4.1. Both settings have s set to 0% and f to 1.5. They differ in their window size. The first setting uses a window size of 4910 for the compression. In Figure 4.2a, its resulting reconstructed series is shown. The second setting (see Figure 4.2b) has a window size of 10,000. The setting with the larger window size has both a higher compression factor (3984 instead of 3652) and a smaller nRMSE (0.09 instead of 0.15) of the reconstructed series.

First, we take a closer look at the Fourier transform heuristic. Regarding its robustness, there are certain drawbacks. The heuristic calculates a low-pass filtered series and measures how far the original series deviates to it. In the case of the Fourier transform heuristic, it depends highly on the time series how near a low-pass filtered series is to it. When only a few sine waves influence the series' broad appearance, the low-pass filter works as expected (see Figure 4.1b). However, when this is not the case, there can be larger deviations to the original time series in the low-pass filtered series (see Figure 4.1a). Therefore, sometimes a smaller window size generates a larger ε, while a significantly higher window size results in a smaller ε. This results in compression settings, where the nRMSE actually decreases, when the window size is highly increased (see Figure 4.2). When the user increases the window size, they normally do not expect such a behavior. Therefore, the Fourier transform heuristic may not be a suitable choice to them.

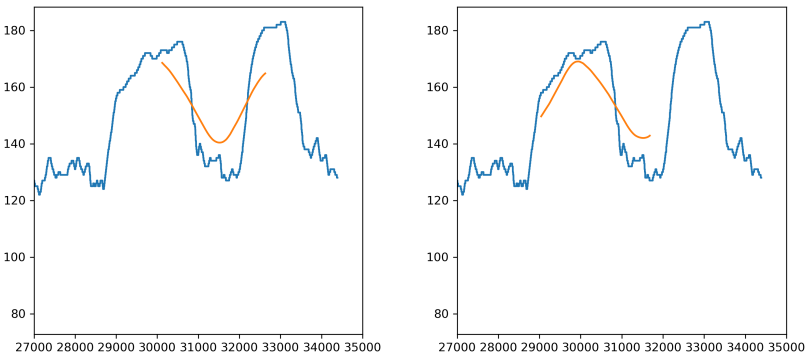
When we look at the heuristic that computes a regression line to calculate ε, we run into a related issue. The reason for this is that the line's fit to the series highly depends on the window. For instance, in a window with a persistent slope, the regression line follows the series quite closely. An example for such a case can be seen in Figure 4.3b . In these segments, we compute a small ε. However, if in the beginning, the series has a steady decrease and afterwards a steady increase, it can not be approximated with a regression line anymore. In figure 4.3a, it can be seen that when the window is slightly altered compared to Figure 4.3b, the regression line looks distinctly different. In regards to the robustness of the heuristic, slightly changed parameters can therefore lead to very different calculated ε. This is something that may not be wanted by the user.

The CSMA heuristic and the min-max heuristic might not suffer from this situation as much. The CSMA heuristic computes the CSMA on the whole time series. When the window size is slightly increased, more points are taken into account for the average of a point. Thus, the CSMA line increasingly displays the long term trend instead of single movements. Compared to the linear regression heuristic, the change in the CSMA line happens more gradually however. Therefore, when the parameter settings change slightly, the moving average lines tend to have a similar fit to the original series. In those cases, the calculated parameters ε are therefore often similar. An example for this is visualized in Figure 4.4. The same argument can be made for the min-max heuristic.



(a) (b)

Figure 4.3: A comparison of the regression lines that are calculated by the linear regression heuristic. In Figure 4.3a, the regression for a window of size 2510 is shown. In Figure 4.3b, the regression for a window of size 2640 is displayed. Despite the smaller window size, and despite the fact that both windows are largely overlapping, the regression line for the smaller window has a considerably larger maximum deviation to the original series (36.3 instead of 21.6).



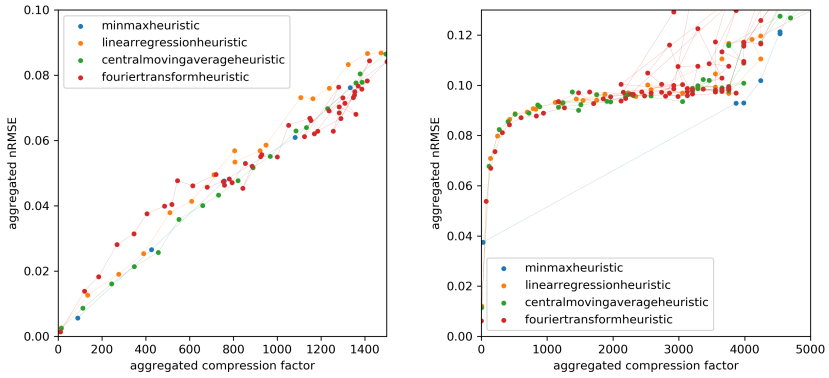
(a) (b)

Figure 4.4: A comparison of the central moving averages that are calculated. Compared to Figure 4.3, the same window sizes and windows are considered. In Figure 4.4a, the CSMA with a window size of 2510 is shown. Figure 4.4b displays the CSMA with a window size of 2640. In this case, both calculated maximum deviations are similar, with 17.2 for the smaller window and 19.7 for the larger one.

Now that we discussed certain properties of the heuristics, we show them experimentally. For that, we look at the Pamap2 dataset as a representant of a dataset with only few oscillations. To represent a dataset that has many variations in a short window, we consider the Beijing $PM_{2.5}$ dataset as well. We take every time series of those datasets into account and a large number of possible window sizes. To evaluate the effect of an increasing window size, the scaling factor f and the parameter s, which denotes the percentage of neighboring windows taken into account per window, are set to a fix value. To better examine the effect of varying partitions of the windows, we set s to 0%. We set f such that when the original series has many fluctuations, it is possible for them to not be present in the compressed series. With this criterion, we determined f to be equal to 1.5 empirically. For each dataset, we generate 100 data points. Each of them consists of a different window size and the two fix parameters. We choose the different window sizes such that they are equidistantly spread over the length of the time series. For each point, we calculate the aggregated compression factor and aggregated nRMSE. Figure 4.5 shows the results.

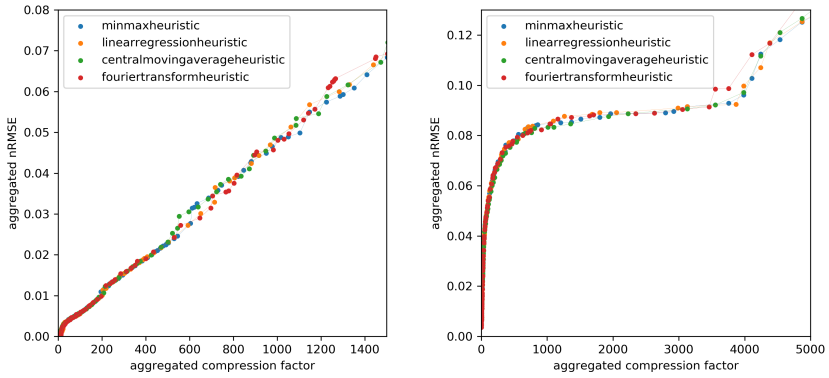
There are a few things to note here. On the Pamap2 dataset (see Figure 4.5a), the linear regression heuristic performs worse than the CSMA and min-max heuristic. For any aggregated compression factor, the resulting error of the linear regression heuristic is higher than for these other two heuristics. This can be due to the fact that it calculates large deviations ε in important areas, when the windows are unfavorably split. Since there are multiple series in the dataset, it is likely that for any window size, there exists one time series where the division of windows is disadvantageous. With the Fourier transform heuristic, we see that an increasing window size often leads to a decreasing aggregated compression factor and nRMSE, which we suspected to happen. The second experiment on the Beijing $PM_{2.5}$ dataset (see Figure 4.5b) demonstrates that the regression heuristic is not necessarily less robust than the CSMA heuristic. In the case of this dataset, with the highly fluctuating movement of its series, the regression line and CSMA line tend to look similar per window. However, again we see that the Fourier transform heuristic does not perform robustly.

From the analysis and from the experiments, we conclude that regarding robustness, the Fourier transform heuristic does not show very promising results. Furthermore, there are time series where the computed deviations ε for the windows are inconsistent, when the linear regression heuristic is used. This problem is less severe, when the CSMA heuristic and min-max heuristic are



(a) (b)

Figure 4.5: In Figure 4.5a the results for the Pamap2 dataset are displayed. Figure 4.5b shows the results for the Beijing $PM_{2.5}$ dataset. Each Figure shows for each different heuristic, the development of the aggregated nRMSE and aggregated compression factor. However, the parameter settings are restricted for this calculation. The parameter s is fixed at 0% and f is fixed at 1.5. The window sizes are chosen equidistantly over the length of the series. A point for a window size is connected with a line to the one for the next larger window size.



(a) (b)

Figure 4.6: For the same datasets as in Figure 4.5, the development of the aggregated nRMSE with the aggregated compression factor is shown for each heuristic. This time, per heuristic the parameters are optimized for it. Figure 4.6a shows the result for the Pamap2 dataset, Figure 4.6b the results for the Beijing $PM_{2.5}$ dataset.

used.

General results

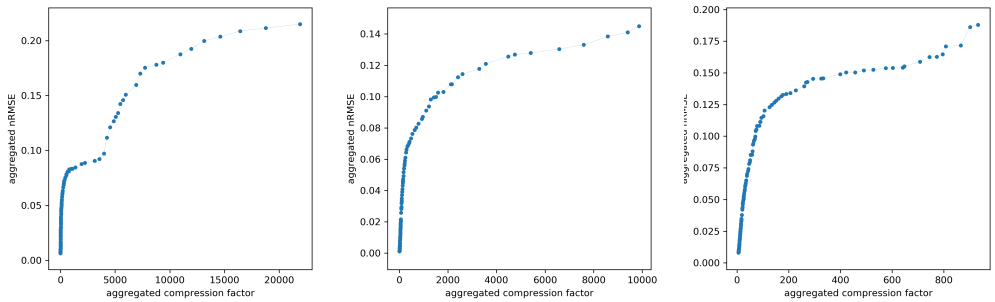
But what are the results when the three parameters get optimized? Do the more robust heuristics perform better in this regard as well? For this, we generate non-dominated parameter choices for each heuristic with the NSGA2 algorithm. The results of this experiment are shown in Figure 4.6. When the parameters are optimized instead of fixed at a certain value, the heuristics perform remarkably similar. As a consequence, we concentrate on the results of the CSMA heuristic in the upcoming sections, since this heuristic is quite robust with respect to changes to parameters such as the window size.

4.2 Performance on Different Datasets

For each dataset, we run our compression algorithm on every selected time series. Using the NSGA2 algorithm, we perform a multi-objective optimization of the parameters, with the aggregated nRMSE and aggregated compression factor as objectives. Then, we analyze the generated pareto fronts. For the analysis, we partition our datasets into two categories. The first category contains the datasets where its series have a large amount of variations in a small window, while the second dataset contains the datasets where the amount of variations is rather small in their series.

To split the datasets by their amount of variations, we could generate anecdotal evidence by looking at some of their time series. However, we also may be able to quantify the amount of variations characteristic for a dataset. To approximate this, we compute how similar the dataset's series are to a central simple moving average of them. The more variations a series has, the less similar it is to a moving average of it (more on this can be found in Section 4.4). Then, we can, for a dataset, aggregate the obtained distances to the moving average and use this as a measure for the amount of variations present in it. To compute this, we need to set a sensible window size w for the moving average. We choose w large enough, such that the variations in the series are filtered by the moving average. However, if every movement is filtered out by it, even series with a low amount of variations may have a low similarity to the moving average. To satisfy these requirements, we look, for every time series T, at the moving average with w set to $\frac{1}{50} \cdot |T|$. The number $\frac{1}{50}$ was generated experimentally on our datasets. However other factors (for instance, $\frac{1}{25}$, $\frac{1}{100}$, etc.) could have been chosen just as well to meet the requirements and bring similar results. To aggregate, for the series in a dataset, the distances to the moving average, we calculate for each of them the nRMSE to the moving average and afterwards compute the mean over these calculations.

As a result of these calculations, the Beijing $PM_{2.5}$ dataset yields an aggregated nRMSE of 0.08, the Energy Production dataset an aggregated nRMSE of 0.07, and the Human Activity Recognition dataset an aggregated nRMSE of 0.08. For the other two datasets, the Pamap2 dataset and the Kinect dataset, the aggregated nRMSE is equal to 0.02 in both cases. Because of this significant difference, we count the Beijing $PM_{2.5}$, Energy Production, and Human Activity Recognition datasets to the datasets with a large amount of variations and the other two datasets to the datasets that have a smaller amount



(a) (b) (c)

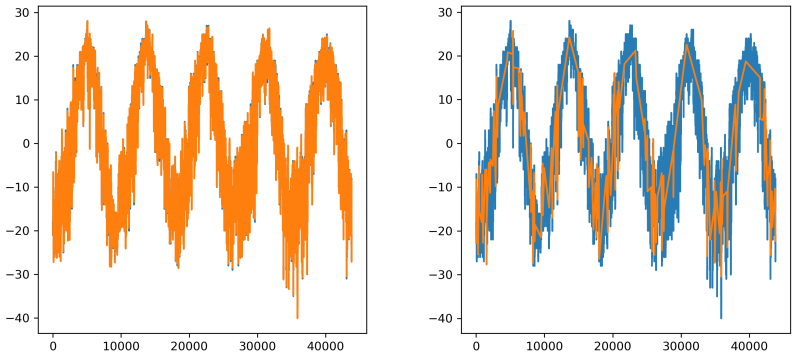
Figure 4.7: Pareto fronts for different datasets. For each dataset, the CSMA heuristic is used and the parameters w, s, and f are optimized for it. The pareto fronts show, for the non-dominated parameters, the development of the aggregated nRMSE and aggregated compression factor. Figure 4.7a shows the pareto front for the Beijing $PM_{2.5}$ dataset, Figure 4.7b the one for the Energy production dataset, and Figure 4.7c the pareto front for the Human Activity Recognition dataset.

of variations.

Large Variations

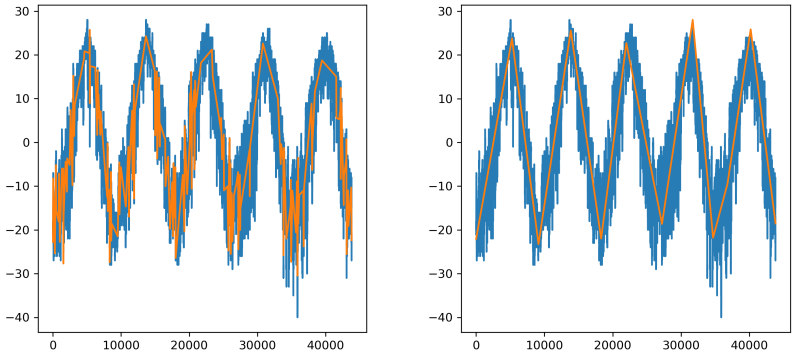
In Figure 4.7, for the datasets with a large amount of variations, the pareto fronts for the aggregated error in dependence of the aggregated compression factor are shown. In the beginning of each front, an increase of the aggregated compression factor leads to a massive increase in the aggregated error. For larger compression factors, the error only increases marginally. Why is that so? To have a reconstructed series with a small nRMSE of, say, 0.01, it needs to contain the majority of the original series' variations. To keep them, however, leads to a small compression factor (see Figure 4.8a). When the compression factor is raised, the compression algorithm draws a straight line through most of the variating points (see Figure 4.8b), which results in a high nRMSE. This is a general issue for every series that has a high number of variations. When the user wants to keep the variations, they have to accept a low compression factor.

However, if they do not need to have the variations, for example because they are considered noise in the application domain, the achievable compression factor increases drastically. An example for such a compression is shown in Figure 4.9b. To only keep some of the variations in this case may not be sensible from the perspective of the user, but also it is not ideal when we look



(a) (b)

Figure 4.8: Two compression settings are compared. The reconstructed series of the first compression setting ($w = 11$, $s = 100\%$, $f = 1.00$) is shown in Figure 4.8a. The second compression setting ($w = 109$, $s = 100\%$, $f = 9.9$) leads to the reconstructed series displayed in Figure 4.8b. For the first compression setting, the nRMSE of the reconstructed series is still small with 0.006, however the compression factor is small as well with 4.1. For the second setting, in which the compression factor is, with 276, not as small, the resulting error is quite large, with 0.07.



(a) (b)

Figure 4.9: Two compression settings are compared. The reconstructed series of the first compression setting ($w = 15$, $s = 100\%$, $f = 9.9$) is shown in Figure 4.9a. The second compression setting ($w = 109$, $s = 0\%$, $f = 4.57$) leads to the reconstructed series displayed in Figure 4.9b. The compression factor for the second compression setting is considerably higher (3652 instead of 276), while the nRMSE of the reconstructed series is not increased by much (0.09 instead of 0.07).

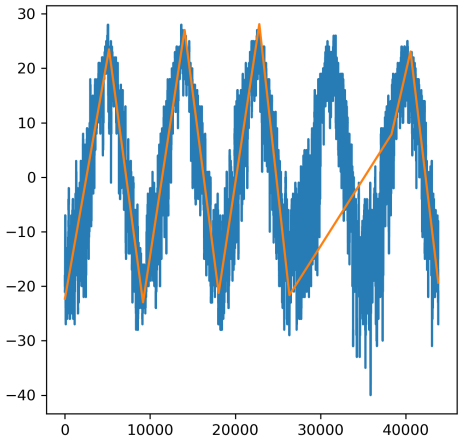
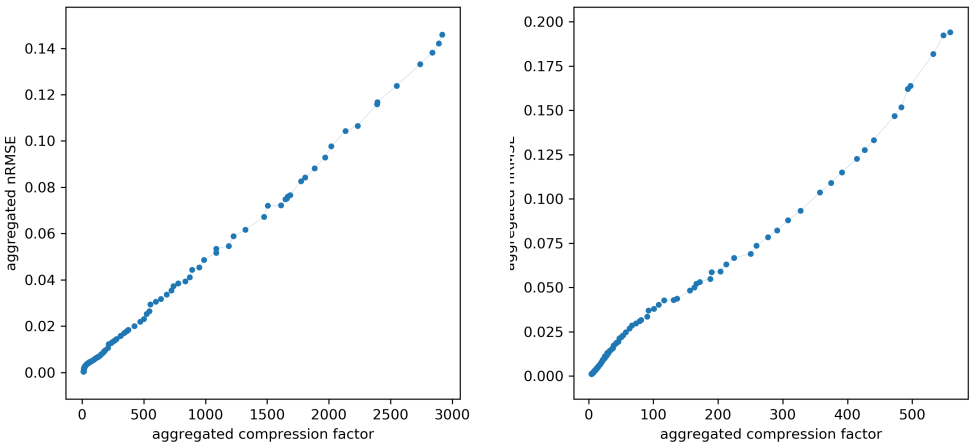


Figure 4.10: A reconstructed series is shown, which skipps some seasonal patterns. To generate this series, w was set to 58, s to 0%, and f to 13.8. Compared to the compression displayed in 4.9b, the nRMSE of the reconstructed series increases significantly (0.16 to 0.09), while the compression factor increases less dramatically (4382 instead of 3652).

at the objectives. In Figure 4.9a, the reconstructed series still contains some variations. The nRMSE for this example is only slightly smaller than on the right figure, while its compression factor is decreased by 13 times. For settings with a high amount of variations, there are two advisable options for the user in most cases. Either they keep every variation, or keep none of them.

When we compare the pareto front of the Beijing $PM_{2.5}$ dataset to the pareto fronts of the Energy Production dataset and Human Activity Recognition dataset, there is an interesting difference between them. After the pareto front for the Beijing $PM_{2.5}$ dataset flattens, it displays a significant increase again for aggregated compression factor larger than 5000. On the other hand, the front for the Energy production and Human Activity Recognition datasets do not show such a salient point. This difference can be explained by the seasonality of the series. As we discussed earlier, every series in the Beijing $PM_{2.5}$ dataset has a strong yearly seasonality. Consequently, there is a stark increase in the nRMSE of the reconstructed series, when even one of the seasonal patterns is skipped due to the compression (see Figure 4.10). On the other hand, when those longer seasonal patterns are not present in a series, the nRMSE in dependence of the compression factor does not show such a bend.



(a) (b)

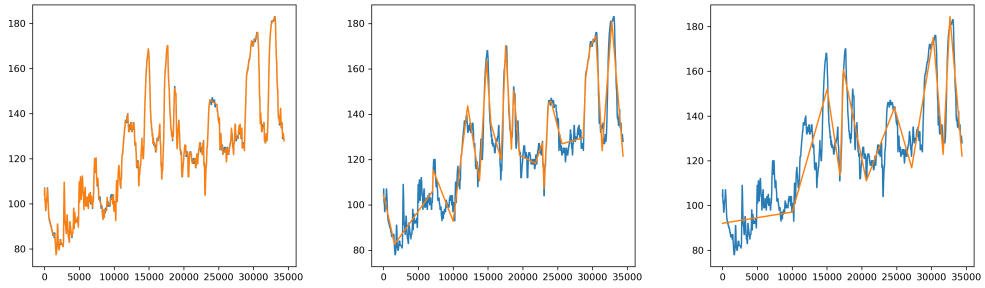
Figure 4.11: The pareto fronts are created similarly as in Figure 4.7. Figure 4.11a displays the pareto front for the Pamap2 dataset, Figure 4.11b shows the one for the Kinect dataset.

To conclude, we see that when a time series has a large amount of variations, there are only two sensible options in most cases. Either the user keeps those fluctuations, or they discard them. Keeping only some of the fluctuations results in a comparatively small compression factor and high nRMSE for the compression of a series. The same can be said, when the series consists of simple seasonal patterns or cycles. Again, if the user decides to keep only some of the patterns, both, compression factor and error, are suboptimal.

Small Variations

When we look at the Pamap2 dataset and the Kinect dataset, they share the similarity that in a short window, their time series do not contain a large number of variations. This property sets them apart from the other datasets. The pareto fronts generated for those datasets display distinct differences (see Figure 4.11). The main difference is that, for non-dominated parameter settings, the aggregated nRMSE scales linearly with the aggregated compression factor. For those datasets, the algorithm achieves a aggregated compression factor of more than 50, while keeping the aggregated nRMSE below 0.01.

We explain the linear movements using an example series of the Pamap2 dataset, which can be seen in Figure 4.12. In the series, there are some areas,



(a) (b) (c)

Figure 4.12: One example time series of the Pamap2 dataset, with three different compression settings. Here, the error increases linearly to the compression factor. The reconstructed series of the first setting ($w = 96$, $s = 100\%$, $f = 3.0$) is displayed in Figure 4.12a. It has a nRMSE of 0.005 and the setting a compression factor of 127. The second setting ($w = 20$, $s = 0\%$, $f = 49.8$, seen in Figure 4.12b) yields a nRMSE of the reconstructed series of 0.05 and a compression factor of 1273. The third setting ($w = 132$, $s = 0\%$, $f = 50.0$, seen in Figure 4.12c) results in a nRMSE of the reconstructed series of 0.087 and a compression factor of 2866.

where it has an upwards trend, without a lot of deviation from it. Then, there are some smaller, variating movements. Furthermore, there are some significant highs in the series, where afterwards, the values drop steadily again. For such a series that does not only consist of variations or seasonal movements, different compression factors result in considerably different errors of the reconstructed series. This can be well seen in the example series. With a small compression factor, every small movement can be captured. In Figure 4.12a it is visible that as a result of this, the error is small as well. For a compression factor 10 times as large compared to the one in Figure 4.12a, they can not be kept anymore, but at least some of the movements with larger swings can. We see in Figure 4.12b that the error therefore increases noticeably as well, in fact the increase is also tenfold. With an even greater compression factor, even some of these movements are not covered in the series anymore, however, the trend can still be displayed. Therefore, the error rises almost twofold again, when the compression factor increases by 2.25 times. This can be seen in Figure 4.12c.

When we compare the pareto front of the Pamap2 dataset to the one of the Kinect dataset, they look quite similar to each other. Both show a nearly linear scaling. However, the pareto front of the Kinect dataset shows a starker increase, beginning at an aggregated compression factor of 300. One possi-

ble explanation for this phenomenon are the longer seasonal patterns that are contained in some series of the dataset. As described earlier, it can lead to a starker increase in error, when seasonal patterns are not approximated well anymore. Since the dataset consists of a large number of time series, it is possible that this is not the only influence on the rise of the aggregated error with larger compression factors.

To sum it up, it seems like the error's stark increase in the beginning is indeed caused by the variations. In datasets, in which these are only present to a small degree, the aggregated error increases linearly with the aggregated compression factor. For this reason, the user can freely choose a compression factor depending on their use case. There are no settings that would be assumed as unattractive beforehand.

4.3 Combinations of Parameters

As we laid out, there are three parameters for the algorithm, the size w of the window, the relative amount of windows over which ε is calculated s, and the scaling factor f. When we optimize the parameter for the pareto front, we get a set of parameter choices that yield non-dominated results. In this section, we discuss whether there are patterns in this set of choices.

Windows Around s

In the setting, ε gets calculated per window of the series, which allows the user to have different maximum deviations of the reconstructed series in different areas. The relative amount of neighboring windows to take into account is set by the parameter s. We investigate whether there are datasets, for which the values of s are similar to each other within the non-dominated parameters.

To evaluate this, we consider the pareto fronts that we generated in Section 4.2. For these fronts, we used the aggregated nRMSE as error measure. In addition to this, we create two new pareto fronts for each dataset, one with the aggregated nMAE as error measure, and one using the aggregated nL_∞ error. For each pareto front, the CSMA heuristic is used to calculate the local ε. Per pareto front, we eliminate some of the non-dominated parameter combinations. For the datasets with large variations (Beijing $\mathrm{PM}_{2.5}$, Energy Production, Human Activity Recognition), we saw that in their pareto fronts, there is an area in the beginning where the error shows a stark increase. This

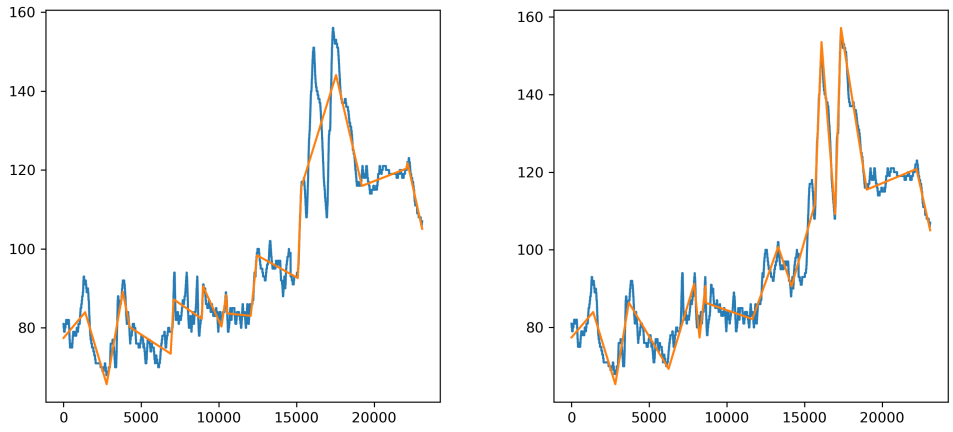
Dataset	nMAE	nRMSE	nL_∞
Pamap2	54.2%	21.6%	1.7%
Kinect	25.7%	10.1%	1.6%
Beijing $PM_{2.5}$	94.7%	97.7%	14.6%
Energy Production	87.5%	65.0%	13.5%
Human Activity Recognition	58.3%	28.4%	9.4%

Table 4.1: We observe per combination of dataset and error measure the relative amount of non-dominated parameter choices, in which s has a small value.

is not only the case for the pareto fronts with the aggregated nRMSE as error measure, but also for the pareto fronts with aggregated nMAE and aggregated nL_∞ as measures. Since we presume this may not be attractive to the user, we exclude parameter settings whose results are located in this part of the pareto front. To very roughly approximate this, we only consider per experiment on these three datasets the non-dominated parameter settings, where the respective aggregated error is larger than 0.05. Additionally, we cap the maximum allowed error for all datasets. For this, we look at a reconstructed series that only consists of a constant value, which is the original series' mean value. When only the mean value is stored as a compression of the series, the resulting compression factor would be larger than any compression factor achievable with the Ramer-Douglas-Peucker algorithm. If the aggregated error of storing the mean is smaller than the aggregated error of a parameter combination, we can therefore disregard the parameter combination.

With these restrictions, we observe per dataset and error measure how often a non-dominated parameter combination has a small value for s. We denote s as small, when it is less than 0.025. Since neighbors on both sides of the window are considered, this means at most 5% of all windows are taken into account to calculate the local ε. The results can be seen in table 4.1.

The Beijing $PM_{2.5}$ dataset and the Energy Production dataset consist of series with larger variations that do not appear seasonally, and outliers. The Human Activity Recognition dataset consists of many variations as well, however they occur seasonally and not irregularly. The Kinect dataset and the Pamap2 dataset contain only a few fluctuations in a short window. For the datasets with a lot of irregular variations, the optimizations for the aggregated nRMSE and aggregated nMAE often result in parameter combinations where s is small. For datasets that contain only a few variations, or seasonal variations, the non-dominated parameter combinations do considerably less often consist

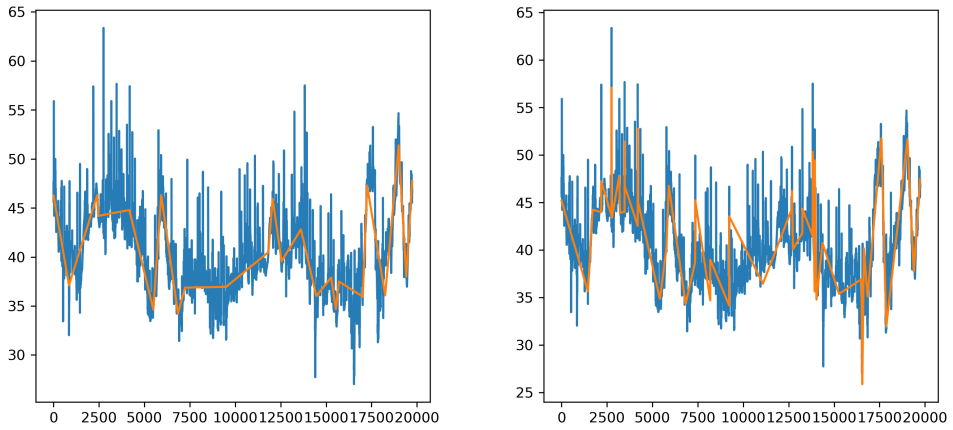


(a) (b)

Figure 4.13: One example time series of the Pamap2 dataset. For both results of the compression algorithm, window size and the scaling factor are the same. In the left figure, no window in the neighborhood is considered. In the right figure, we take the median over all windows. The latter approach result in a higher compression factor (1215 instead of 1100) and a 40% lower nRMSE (0.037 instead of 0.058) of the reconstructed series.

of a small value for s. Furthermore, we observe that the optimizations for the aggregated nMAE yield more often non-dominated parameter combinations with a small parameter s, than the optimizations for the aggregated nRMSE. Optimizations for the aggregated nL_∞ error do not often yield non-dominated combinations with a small s.

To illustrate the effect of a small value for the parameter s, we view the most extreme case, in which s is equal to 0%. In this case, windows in which the values deviate largely from the CSMA generally have a larger ε than windows where this is not the case. In a series that has areas where a stark increase is followed by a stark decrease, these areas can therefore potentially be omitted in the compressed series without violating the bound ε. An example for this can be seen in Figure 4.13a. If we medianize the windows' deviations to the CSMA, points in these areas have a smaller bound ε than without medianizing. Consequently, they are represented in the reconstructed series. This can be seen in Figure 4.13b. In this case, the error with medianizing is smaller than without it, even though other areas of the series are less closely approximated by the reconstructed series.



(a) (b)

Figure 4.14: One example time series of the Energy Production dataset. Again, in the left figure we consider no neighboring windows, while we take the median over all windows in the right figure. The window size w and scaling factor f are set to a fix value value. To consider no neighboring windows results in a compression factor 2 times higher (822 instead of 411), while the nRMSE of the reconstructed series is 14% smaller (0.058 instead of 0.067).

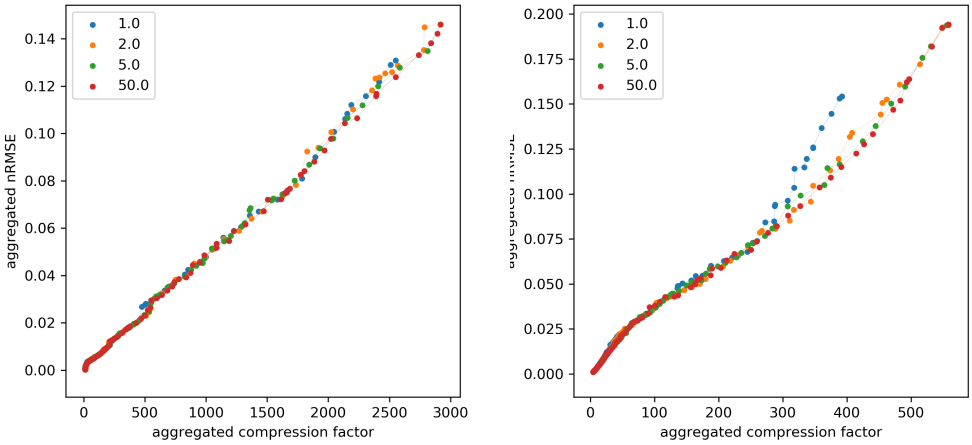
However, not every series has as few oscillations as the series in the Pamap2 dataset. The Human Activity Recognition dataset contains many variations, however it has no outliers or much irregular movement in general. There do however exist series that consist of outliers. These are partially present in the Beijing $PM_{2.5}$ dataset and even more so in the Energy Production dataset. With the original Ramer-Douglas-Peucker algorithm, the user needs to set a large ε to get a compressed series that does not keep these outliers. This way however, interesting movements in other areas may get lost. In those cases, a flexible setting of ε comes in handy. Now, the outliers are covered by the sizable ε for their respective windows. Additionally, areas with more steady movements are taken into consideration, because of a smaller ε in their respective windows. In Figure 4.14, we see how the compression can be improved in regards to the nRMSE this way. To conclude, we see that with a small value for the parameter s, the algorithm is able to take outliers less into account. We saw in the experiment that the non-dominated parameter choices depend on the error the parameters are optimized for. It showed for the aggregated error measures that the nMAE has the largest amount of combinations with a small s, while the nL_∞ error has the fewest by far, with the result for the nRMSE being somewhere in between them. This can be explained by the error mea-

sures' punishment when outliers are not kept in the reconstructed series. The less severe the error function punishes this, the more often a non-dominated parameter combination leads to a reconstructed series that does not keep outliers, and therefore the parameter combination often has a small value for s. In conclusion, the parameter s can be a tool for the user to indicate that the reconstructed series should include more or less outliers.

Relation Between Scaling Factor and Window Size

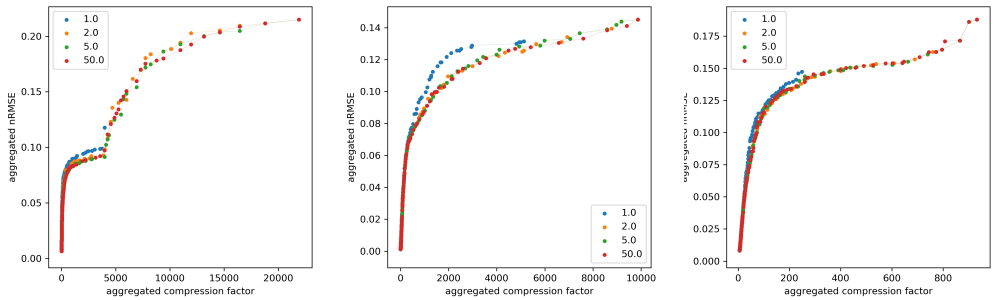
The second parameter is the scaling factor. For every window, an ε is calculated. After the calculation, the user is allowed to scale ε by a factor f. Therefore, in the compression setting there are two ways to increase the compression factor of a compressed series. Either heighten the window size, or enlarge the scaling factor. It could be suspected that the result of a larger scaling factor can also be achieved by a heightened window size. If this was the case, we would only need the window size w as a parameter and could disregard the scaling factor. Therefore, we conduct experiments to evaluate whether this assumption holds true. In order to do this, we change the upper limit the scaling factor can adopt. In the experiments, this limit was 50. Now, we consider the limits 5.0, 2.0, and 1.0 as well. For each limit and dataset, we conduct the experiment and generate a pareto front. We use the CSMA heuristic to calculate ε. Also, we use the aggregated nRMSE as the error measure. Note that the lower bound of the scaling factor is always 1.0. Thus, we see whether it influences the compression when the factor has a smaller range. Additionally, we examine whether the scaling factor could be fixed at 1.0.

As we can see in the resulting pareto fronts, it does have indeed no measurable effect, whether the limit is set to 50, 5 or 2. The very slight variations can be explained by the way the NSGA2 algorithm works. Since it does not try out every combination, the pareto fronts are only approximated. Therefore, slight differences in the fronts do not have any explicative value. However, there is an emerging pattern. In the Kinect dataset, the Beijing $PM_{2.5}$ dataset, and the Energy Production dataset, the compression algorithm performs worse when the factor is fixed at 1.0, especially for larger aggregated compression factors. This is displayed in Figure 4.15 and 4.16. When we consider our results of the previous section, we observe a certain correlation. As a reminder, in the previous section we tested on which datasets we gain anything when we have a lot of differing ε in one time series. Those were mainly comprised of the Beijing $PM_{2.5}$ dataset and the Energy Production dataset. In those datasets, with a fix scaling factor of 1.0 the compression algorithm performs worse in regards



(a) (b)

Figure 4.15: The resulting pareto fronts that display the development of the aggregated nRMSE with the aggregated compression factor for the non-dominated parameter settings. For each pareto front, the scaling factor f is limited to a different maximum value. Figure 4.15a displays the results for the Pamap2 dataset and Figure 4.15b the results for the Kinect dataset.



(a) (b) (c)

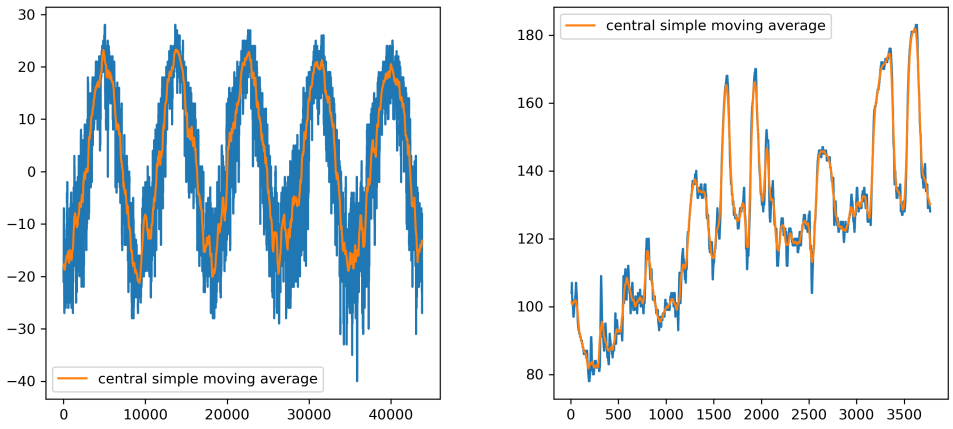
Figure 4.16: The pareto fronts are generated the same way as in Figure 4.15. Figure 4.16a shows the results for the Beijing $PM_{2.5}$ dataset, Figure 4.16b the results for the Energy Production dataset, and Figure 4.16c the results for the Human Activity Recognition dataset.

to multi-objective optimization. One possible explanation for this is related to the diversity of the ε in the windows. When ε vastly differs per window, the partition of the windows can influence the result of the compression. For instance, in the Energy Production dataset, with a larger window size outliers and significant movements can occur in one window. As explained earlier, this can create a dilemma. Either we choose a small ε to depict the movements, or choose a large ε to omit the outlier. In those cases, we can increase the scaling factor f to increase the compression factor, while the partition of the windows stays the same. In the Kinect dataset, with a fix f, the compression algorithm performs worse for aggregated compression factors larger than 300. This is illustrated in Figure 4.15. In the dataset, for aggregated compression factors larger than 300, 87.5% of the non-dominated parameter combinations have a small value for s as well. Therefore, its result is in line with the results for the $PM_{2.5}$ dataset and the Energy Production dataset. In the other datasets, a fixed scaling factor yields nearly equal results. We can assume that in those cases, the partition of the windows matters less, since for many of the non-dominated parameter combinations, every window has the same value for ε.

Summing up, we see that the scaling factor and the flexibility of ε allow the user additional fine-tuning. With both parameters combined, the user can compress the series such that the outliers are rather not kept, while other, more consistent movements are displayed.

4.4 Window Size

Now, we discuss how we can use the window size to approximate the pareto front. As we argued before, the window size is a powerful tool to adjust the compression setting. Even though the scaling factor yields additional value for some datasets, its addition does not dramatically change the characteristics of the pareto fronts (see Figure 4.15, 4.16). Additionally, we observed that depending on the dataset, the pareto fronts look vastly different to each other. When the pareto front has a linear progression, the user has many options for the window size. If, however, the pareto front displays a stark increase of the error for small aggregated compression factors, a small window size is normally not advisable. Therefore, the suitable parameters depend on the characteristics of the pareto front. It would thus be beneficial if the user knows beforehand what front they can expect for a dataset. In this section, we present a simple procedure that allows them to do this. When we use the CSMA heuristic, we



(a) (b)

Figure 4.17: Two time series are compared. The left figure shows a time series of the Pamap2 dataset, while the right figure display a series of the Beijing $PM_{2.5}$ dataset. For both series, the moving average per point is calculated for a window size w of 600. In the left figure, the effect of the smoothing is more significant.

calculate for a window size w the central moving average of a time series. For each w, the value per point is calculated over $\frac{2}{3} \cdot w$ values. For the approximation, we calculate the moving average with multiple window sizes w. Then, we plot the dissimilarity of those moving averages to the original time series in dependence of their window size.

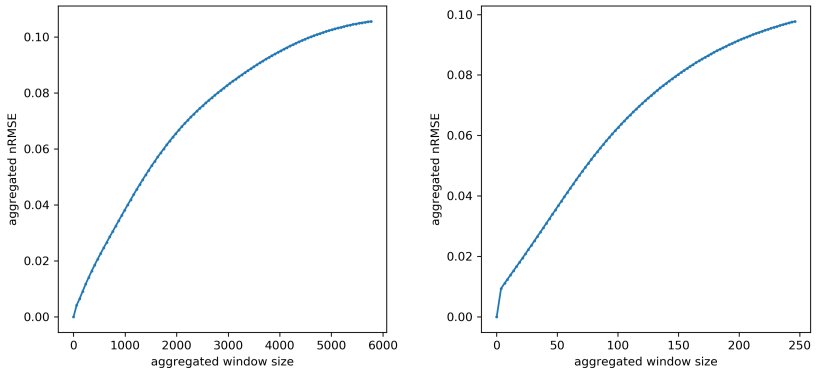
The idea behind this approach is to approximate the error that is caused by the variations and seasonal movements. We can consider a scenario, in which the series is smoothed with a moving average that averages only over a small amount of values. When the series has many oscillating movements, the average already smoothes them out (see Figure 4.17a), which leads to a large difference of the moving average to the original series. However, a series with slower, more consistent changes is not altered significantly when smoothed this way (see Figure 4.17b) and thus the moving average is close to the original series in this case. The larger w is, the more values are taken into account to calculate the value for each point. Therefore, a greater w leads to a moving average that further follows the long term trend.

For each calculated moving average, we calculate its difference to the original series with an error measure. Similarly as for the compression, we use

the nRMSE. We calculate, per series, for multiple w the error of the moving average with w as window size. More precisely, we use 100 equidistant window sizes per series. Finally, we need to aggregate the calculated errors and window sizes over the entire dataset. To do that, we look for each $i \in [1, 100]$ at the i-th smallest window size of every series. Then, we calculate the mean over each i-th smallest window size. In addition, we calculate the mean of each i-th smallest size's resulting error. The resulting point then consists of an aggregated window size and its aggregated error. While we conducted the experiments, we detected that when the window size is set to a significant portion of the series' length, there are not much changes in the moving average anymore. This holds true for every dataset on which we conducted the experiment. Therefore, we slightly limit the range over which we sample the window sizes. For each series T, we set the range to $[0, |T|/5]$.

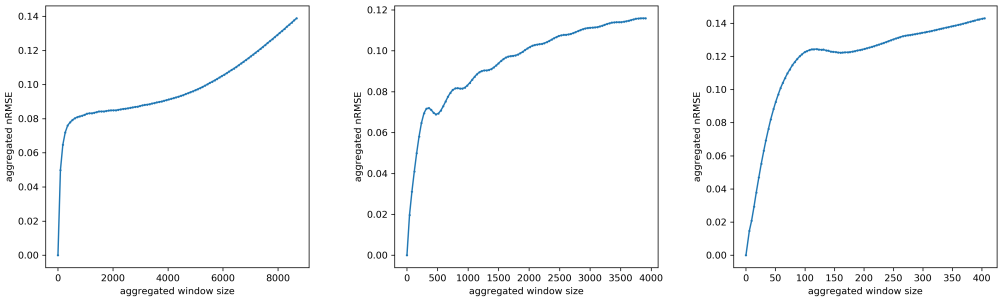
Now, we analyze whether the approximations that we calculate this way are similar to the pareto fronts. In the previous experiments, we detected that the pareto fronts for the Pamap2 dataset and Kinect dataset dataset showed linear progression. For these datasets, the moving average approximations can be seen in Figure 4.18. We see that they do indeed display a roughly linear scaling as well. For the Beijing $PM_{2.5}$ dataset, the Energy Production dataset, and the Human Activity Recognition dataset, the pareto fronts showed a considerable increase in error for smaller aggregated compression factors. In Figure 4.19 it is visible that for the calculated approximations, the growth in error for smaller aggregated window sizes w corresponds to this. In the approximation for the Beijing $PM_{2.5}$ dataset, the aggregated nRMSE for larger aggregated window sizes increases again, after slowing down. Again, this is behavior present in the actual pareto front as well.

Concluding, we developed a technique which enables to approximate the pareto front for a dataset. Although the approximations do not look exactly the same as the pareto fronts, they preserve their main characteristics. With this, the user is able to get information on the behavior of the compression algorithm without compressing the series. Especially when the user only samples some time series of a dataset and some window sizes, the approximation can be calculated in a few seconds for the datasets in this work.



(a) (b)

Figure 4.18: The pareto fronts are approximated with the central moving average. For this, the aggregated nRMSE of the central simple moving average is plotted against its aggregated window size. In Figure 4.18a the front of the Pamap2 dataset is approximated, and Figure 4.18b the front of the Kinect dataset.



(a) (b) (c)

Figure 4.19: The fronts are approximated similar as in Figure 4.18. In Figure 4.19a, the front of the Beijing $PM_{2.5}$ dataset is approximated, in Figure 4.19b the front of the Energy Production dataset, and in Figure 4.19c the front of the Human Activity Recognition dataset.

5 Conclusion

When a time series needs to be compressed, the extent of the compression depends on the use case for the series. In a setting where machines are monitored, there are many different ways imaginable to compress the sensors' values. However, the user needs to find parameters such that the compression fits to their use case.

With this work, we aid the user in the selection of suitable parameters. It is based on the work that was part of the bachelor's project. In the project, we developed a heuristic that computes a tolerable deviation ε for any given time series. To compute this allowed deviation ε, the heuristic needs to calculate how far points deviate from the overall trend. In this work we showed that even though there exist multiple approaches to calculate this, the error in dependence on the compression factor develops very similar between the pareto fronts. Furthermore, we argued that calculating the deviation to the CSMA is very robust, which can not be said for other approaches, such as the linear regression heuristic or the fourier transform heuristic.

We proposed a change to the heuristic that allows it to compute multiple allowed deviations ε per time series. We analyzed whether this change can lead to different compressions by the algorithm. We observed that especially when the series contains outliers, the heuristic can calculate the multiple ε such that the compression algorithm assigns such outliers a lower priority. We further showed that for error measures such as nMAE and nRMSE, this can yield better results.

For the compression of a time series, we demonstrated that it depends on the data how the error develops for increasing compression factors. When a series has a lot of variations, the error is already high for small compression factors. When this is not the case, the error develops slowly in the beginning. For the user this may be interesting, since it shows that for different types of

data, different compression settings could be suitable.

When the user wants to choose parameter settings for a time series depending on its characteristics, they would need to know the characteristics beforehand. In this work, we proposed a way to aid the user with this. For this, we developed a way to approximate the pareto front of a time series' compression. As approximation, we propose to calculate a moving average with different window sizes. Then, we visualize its error to the original series in dependence of the window sizes. With this approximation, the user is able to project the expected pareto front before the compression algorithm starts to compress a time series.

The parameter values that the user can choose, however, could be put to further research. Especially interesting is whether one can estimate the position of a compression setting in the pareto front based on the parameter choice and the characteristics of a dataset. For this, one could research how the calculated error of a moving average with window size w relates to the error of a compression where the heuristic has window size w.

To conclude, in this work we set out to aid the user in regards to the parameter configurations and their effects on the compression. For this, we showed how the user, by the use of the heuristic, is able influence the properties of the compression. We additionally analyzed for different datasets, how the error of non-dominated parameter settings develops with the compression factor. Not only did we show that there are significant differences between the datasets, we further developed a tool that enables the user to spot these before the compression. With the results of our work, the user now has the information and tools available that allow them, for their use case, to achieve a fitting compression for it.

References

[1] Analog filters. URL https://www.analog.com/media/en/training-seminars/design-handbooks/Basic-Linear-Design/Chapter8.pdf. (Cited on page 14.)

[2] https://www.microsoft.com/en-us/download/details.aspx?id=52283&from=http (Cited on pages 18 and 23.)

[3] Regression. URL http://uregina.ca/~gingrich/regr.pdf. (Cited on page 12.)

[4] Electronic statistics textbook. URL http://www.statsoft.com/textbook/. (Cited on page 9.)

[5] A. Anand, L. Wilkinson, and D. N. Tuan. An l-infinity norm visual classifier. In *2009 Ninth IEEE International Conference on Data Mining*, pages 687–692. IEEE, 2009. (Cited on page 16.)

[6] J. Andersson. A survey of multiobjective optimization in engineering design. *Department of Mechanical Engineering, Linktjping University. Sweden*, 2000. (Cited on page 21.)

[7] R. Bellman and B. Kotkin. On the approximation of curves by line segments using dynamic programming. ii. Technical report, RAND CORP SANTA MONICA CALIF, 1962. (Cited on page 2.)

[8] L. M. Candanedo, V. Feldheim, and D. Deramaix. Data driven prediction models of energy use of appliances in a low-energy house. *Energy and buildings*, 140:81–97, 2017. (Cited on pages 19 and 25.)

[9] H. Chen, J. Li, and P. Mohapatra. Race: Time series compression with rate adaptivity and error bound for sensor networks. In *2004 IEEE International Conference on Mobile Ad-hoc and Sensor Systems (IEEE Cat. No. 04EX975)*, pages 124–133. IEEE, 2004. (Cited on pages 1 and 2.)

[10] S. P. Coy, B. L. Golden, G. C. Runger, and E. A. Wasil. Using experimental design to find effective parameter settings for heuristics. *Journal of Heuristics*, 7(1):77–97, 2001. (Cited on page 2.)

[11] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, April 2002. ISSN 1089-778X. doi: 10.1109/4235.996017. (Cited on page 21.)

[12] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: the international journal for geographic information and geovisualization*, 10(2):112–122, 1973. (Cited on page 6.)

[13] F. Eichinger, P. Efros, S. Karnouskos, and K. Böhm. A time-series compression technique and its application to the smart grid. *The VLDB Journal—The International Journal on Very Large Data Bases*, 24(2): 193–218, 2015. (Cited on page 2.)

[14] H. Elmeleegy, A. K. Elmagarmid, E. Cecchet, W. G. Aref, and W. Zwaenepoel. Online piece-wise linear approximation of numerical streams with precision guarantees. *Proceedings of the VLDB Endowment*, 2(1):145–156, 2009. (Cited on page 2.)

[15] V. Froese and B. Jain. Fast exact dynamic time warping on run-length encoded time series. *arXiv preprint arXiv:1903.03003*, 2019. (Cited on page 1.)

[16] D. Ganesan, B. Greenstein, D. Perelyubskiy, D. Estrin, and J. Heidemann. An evaluation of multi-resolution storage for sensor networks. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 89–102. ACM, 2003. (Cited on page 1.)

[17] P. Heckbert. Fourier transforms and the fast fourier transform (fft) algorithm. *Computer Graphics*, 2:15–463, 1995. (Cited on pages 14 and 15.)

[18] R. J. Hyndman. Moving averages. *International encyclopedia of statistical science*, pages 866–869, 2011. (Cited on page 13.)

[19] A. L. Jaimes, S. Z. Martınez, and C. A. C. Coello. An introduction to multiobjective optimization techniques. *Optimization in Polymer Processing*, pages 29–57, 2009. (Cited on page 20.)

[20] E. Keogh, S. Lonardi, and C. A. Ratanamahatana. Towards parameter-free data mining. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 206–215. ACM, 2004. (Cited on page 2.)

[21] T. Kohler and D. Lorenz. A comparison of denoising methods for one dimensional time series. *Bremen, Germany, University of Bremen*, 131, 2005. (Cited on pages 13 and 14.)

[22] X. Liang, T. Zou, B. Guo, S. Li, H. Zhang, S. Zhang, H. Huang, and S. X. Chen. Assessing beijing's pm2. 5 pollution: severity, weather impact, apec and winter heating. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 471(2182):20150257, 2015. (Cited on pages 9 and 24.)

[23] M. Malekzadeh, R. G. Clegg, A. Cavallaro, and H. Haddadi. Protecting sensory data against sensitive inferences. In *Proceedings of the 1st Workshop on Privacy by Design in Distributed Systems*, page 2. ACM, 2018. (Cited on page 26.)

[24] T. Palpanas, M. Vlachos, E. Keogh, D. Gunopulos, and W. Truppel. Online amnesic approximation of streaming time series. In *Proceedings. 20th International Conference on Data Engineering*, pages 339–349. IEEE, 2004. (Cited on page 1.)

[25] T. Pelkonen, S. Franklin, J. Teller, P. Cavallaro, Q. Huang, J. Meza, and K. Veeraraghavan. Gorilla: A fast, scalable, in-memory time series database. *Proceedings of the VLDB Endowment*, 8(12):1816–1827, 2015. (Cited on page 1.)

[26] G. Quin, B. Pinel-Puysségur, and J.-M. Nicolas. Comparison of harmonic, geometric and arithmetic means for change detection in sar time series. In *EUSAR 2012; 9th European Conference on Synthetic Aperture Radar*, pages 255–258. VDE, 2012. (Cited on page 20.)

[27] A. Reiss and D. Stricker. Introducing a new benchmarked dataset for activity monitoring. In *2012 16th International Symposium on Wearable Computers*, pages 108–109. IEEE, 2012. (Cited on pages 10, 13, and 22.)

[28] M. Shcherbakov, A. Brebels, N. Shcherbakova, A. Tyukov, T. Janovsky, and V. Kamaev. A survey of forecast error measures. *World Applied Sciences Journal*, 24:171–176, 01 2013. doi: 10.5829/idosi.wasj.2013.24.itmies.80032. (Cited on page 16.)

[29] P. Sun, S. Xia, G. Yuan, and D. Li. An overview of moving object trajectory compression algorithms. *Mathematical Problems in Engineering*, 2016, 2016. (Cited on page 2.)

[30] M. Visvalingam and J. D. Whyatt. Line generalisation by repeated elimination of points. *The cartographic journal*, 30(1):46–51, 1993. (Cited on page 2.)

[31] L. Wen, K. Zhou, S. Yang, and L. Li. Compression of smart meter big data: A survey. *Renewable and Sustainable Energy Reviews*, 91:59–69, 2018. (Cited on page 1.)

[32] A. Zahn. Hybrid genetic algorithm for time series compression by pla. July 2019. (Cited on page 8.)