An Adaptive Algorithm for Online Time Series Segmentation with Error Bound Guarantee

Zhenghua Xu Rui Zhang Ramamohanarao Kotagiri Udaya Parampalli

Department of Computer Science and Software Engineering
University of Melbourne
Victoria, Australia
{zhxu, rui, rao, udaya}@csse.unimelb.edu.au

ABSTRACT

The volume of time series data grows rapidly in various applications such as network traffic management, telecommunications, finance and sensor network. To reduce the cost of storage, transmission and processing of time series data, the need for more compact representations of time series data is compelling. Segmentation is one of the most commonly used methods to meet this requirement. Both PLA and PPA are common segmentation methods which divide a time series into segments and use a linear function or a polynomial function to approximate each segment, respectively. However, while most of the current PLA and PPA methods aim to minimize the holistic error between the approximation and the original time series, few works try to represent time series as compact as possible with an error bound guarantee on each data point. Furthermore, in many real world situations, the patterns of the time series do not follow a constant rule such that using only one type of functions may not yield the best compaction.

Motivated by these observations, we propose an online segmentation algorithm which approximates time series by a set of different types of candidate functions (polynomials of different orders, exponential functions, etc.) and adaptively chooses the most compact one as the pattern of the time series changes. A challenge in this approach is to determine the approximation function on the fly ("online"). Thereby, we further propose a novel method to efficiently generate the compact approximation of a time series in an online fashion for several types of candidate functions. This method incrementally narrows the feasible coefficient spaces of candidate functions in coefficient coordinate systems such that it can make each segment as long as possible given an error bound on each data point. Extensive experimental results show that our algorithm generates more compact approximations of the time series with lower average errors than the state-of-the-art algorithm.

Categories and Subject Descriptors

H.2.8 [**Information Systems**]: Database Application

EDBT 2012, March 26–30, 2012, Berlin, Germany.

General Terms

Algorithm

Keywords

Time Series, Segmentation, Approximation

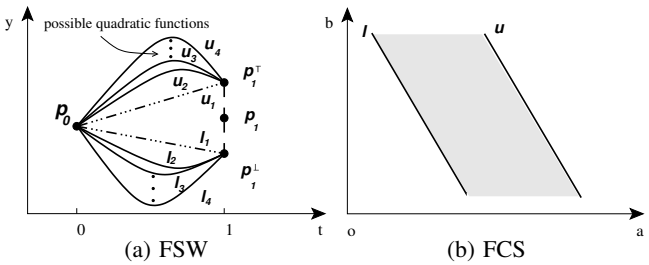
1 Introduction

A time series is a sequence of data points where each data point is associated with a timestamp. There are rapidly increasing research interests in the management of time series data due to its importance in a variety of applications such as network traffic management [13, 33], telecommunications [4], finance [37] and sensor network [6]. These applications need to record the change of certain values (counts of network packets, stock price, temperature, etc.) over time and the recorded values form time series that grow with high speed and continuously. For example:

- In telecommunication, AT&T's call-detail time series contains roughly 300 million calls per day generating approximately 7GBs data each day [7].
- In financial markets, the data vectors like Reuters transmit more than 275,000 prices per day for foreign exchange spot rates alone [5].

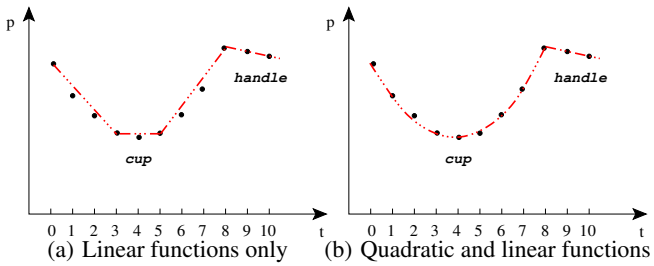
Due to the rapid and continuous growth of data in the above applications, we usually cannot afford to store the entire time series due to the huge volume [2]. This poses a new challenge in data storage, transmission, and processing. Therefore, the need for more compact representations of time series data is compelling. Another important characteristic of the above applications is that the data points in the time series may arrive (or be generated) continually. These applications require continuously monitoring the data and analyzing them in almost real time. Therefore, we need to process the data points and provide answers on the fly, i.e., the algorithms need to be "online".

A common approach to address the problem of the large data volume is *segmentation* which provides more compact representations of time series data through dividing time series data into segments and using a high level representation to approximate each segment. The highly compact segmentation can reduce both the space and the computational cost of storing and transmitting such data, and also reduce the workload of data processing (e.g., more efficient in mining the sequence) [32]. Therefore, in this paper, we try to find a highly compact segmentation scheme that each segment can be approximated by a high level representation given an error bound



(a) FSW (b) FCS

Figure 1: FSW vs FCS



(a) Linear functions only (b) Quadratic and linear functions

Figure 2: Cup and handle in stock price time series

on each data point and the amount of information (i.e., the number of parameters) used to represent the time series is minimized. Furthermore, we require online algorithms in order to accommodate the continuous nature of the data generated in the applications described earlier.

Piecewise Linear Approximation (PLA) [16, 24, 28] has been one of the most widely used segmentation methods for many practical applications [12, 19, 29, 36] because of its simplicity. PLA divides a time series into segments and uses a linear function to approximate each segment. Nonetheless, linear functions may not always be the best choice to approximate a time series due to the different kinds of patterns of the time series. Therefore, *Piecewise Polynomial Approximation (PPA)* [9, 21] is introduced to approximate the time series with polynomial patterns more properly. PPA uses polynomial functions instead of linear functions to approximate segments.

However, the goal of most current PLA and PPA methods [16, 28, 21, 9] is to minimize the holistic approximation error (e.g., the Euclidean distance between the approximation and the original time series) given a certain amount of information (e.g., the number of segments [28] or the maximum error in a segment [16]), where the best approximation result is the one with the lowest holistic error. This goal is different from that of our work: minimizing the amount of information used to represent the time series given a certain error bound on each data point, where the best approximation result should be the one using the smallest amount of information. Therefore, these methods do not satisfy the requirement of our problem and can not be used to solve our problem.

The most recent work that has the same problem setting as ours is the *Feasible Space Window (FSW)* method [24] which introduces the concept of *feasible space* to find the farthest segmenting point of each segment. Feasible space is a space in which any straight line can approximate all the data points read so far with a given error bound on each data point. As the example in Figure 1(a) shows, the area between the boundaries u_1 and l_1 is the feasible space for the approximation of p_0 and p_1. The feasible space is incrementally narrowed when new data points arrive continuously and eventually turns into an empty set at a certain data point so that the previous data point will be the farthest data point that can be approximated by a straight line. Thereby, the FSW can make each approximation line as long as possible and minimize the amount of information used. Since FSW is the state of the art for the problem addressed in this paper, we use FSW as the baseline in the experimental study and describe it in detail in subsection 3.2.

FSW approximates time series by linear functions only. However, in many real world situations, the patterns of the time series do not follow a constant rule. Using only one type of functions may not yield the best compaction. Taking the stock price time series as an example, a typical stock price pattern called *Cup and Handle* [26] has two parts: a "cup" and a "handle" as shown in Figure 2. The cup is a round bottom and it is followed by the handle part which is a straight line. Figure 2(a) shows how this time series is approximated using only linear functions and Figure 2(b) shows how this time series is approximated using quadratic and linear functions. Based on an equation used to calculate the number of parameters (i.e., Equation 2 in Subsection 3.1), the segmentation in Figure 2(a) uses nine parameters to represent the time series while the segmentation in Figure 2(b) only uses six parameters. Therefore, using multiple types of functions may yield more compact approximation.

Motivated by these observations, we propose an online time series segmentation algorithm which approximates time series by a set of different types of functions (such as polynomials of different orders, exponential functions, etc.) and adaptively chooses the most compact one as the pattern of the time series changes. We call this algorithm the *Adaptive Approximation (AA)* algorithm and refer to the functions used to approximate the time series as the *candidate functions*. To achieve our algorithm, we need to solve the following two subproblems: (i) for a candidate function, how to determine the values of its coefficients so that it can approximate as many data points as possible given an error bound on each data point; (ii) from a set of candidate functions, how to determine the one that generates the most compact approximation of a certain part of a time series (a subsequence).

Although feasible space is an intriguing idea for determining the compact approximation of time series data, it is difficult to apply the FSW algorithm to non-linear functions. The idea of FSW is to use a starting point and a next data point to determine the boundaries of the feasible space of the approximation on the fly. Nonetheless, most non-linear functions have more than two coefficients and their approximation boundaries can not be determined by only two data points. As shown in Figure 1(a), two data points p_0 and p_1 can not uniquely determine the upper and lower boundaries of the feasible space for the quadratic function.

We propose a novel method to address this challenge: instead of finding the boundaries of feasible space as FSW does, we find the boundaries of the feasible values for the functions' coefficients – using two data points, we can uniquely identify such boundaries in the coefficient coordinate system. With these boundaries, we can determine a space in which each point is a feasible set of values for the coefficients of the candidate function. In order to distinguish our feasible space from the Feasible Space (FS) in FSW, the feasible space generated in our method is called the *Feasible Coefficient Space (FCS)* and the algorithm used to generate FCS is called the *FCS algorithm*. Taking the quadratic function $y = ax^2 + bx + c$ as an example, a FCS (gray region in Figure 1(b)) in a coefficient space using coefficients a and b as axes is determined through uniquely obtaining the upper boundary u by p_0 and $p_1^\top$ and the lower boundary l by p_0 and $p_1^\perp$.

Besides the FCS algorithm, we further propose an adaptive mechanism to determine the most compact candidate function for each part of a time series (a subsequence). Specifically, given a starting point, we continuously use the corresponding FCS algorithm of each candidate function to segment and approximate the time series until the data point where each candidate function has obtained at least one segment. Then we calculate the numbers of parameters used by the candidate functions to represent the subsequence ending at this data point (denoted as n_p) and choose the candidate function with the smallest n_p as the approximation function of this subsequence.

In summary, we make the following contributions in this paper.

- We propose an online time series segmentation algorithm called the *Adaptive Approximation (AA)* algorithm which approximates time series by a set of candidate functions (e.g., polynomials of different orders, exponential functions, etc.) and adaptively chooses the most compact one as the pattern of the time series changes.
- We propose a novel method called the Feasible Coefficient Space (FCS) algorithm which can efficiently find the farthest segmenting data point for non-linear candidate functions with more than two coefficients (e.g., m^{th}-order polynomials where m is larger than 1). It addresses the drawback of the FSW algorithm, which can only find the farthest segmenting data point for the functions with two coefficients. We also analyze the complexities of the FCS algorithms for various candidate functions.
- We perform an extensive experimental study using both synthetic and real datasets. The results validate the effectiveness of our AA algorithm. It outperforms the state-of-the-art algorithm, FSW, in terms of the number of approximated data points per parameter. At the same time, the AA algorithm usually results in much lower actual errors than those caused by the FSW algorithm given the same error bound.

The rest of this paper is organized as follows. We first review related work in Section 2. Then we provide some preliminaries and a formal problem definition in Section 3. Section 4 presents the FCS algorithms for various candidate functions and Section 5 explains the AA algorithm in detail. Section 6 reports the results of our experimental study. We conclude the paper and discuss future work in Section 7.

2 Related Work

General lossless data compression techniques such as Huffman coding, Lempel-Ziv Codes, etc. [20] can yield data reduction, but they do not exploit the property that two consecutive values of a time series are close. Therefore, they cannot achieve a compaction rate as high as segmentation methods, which are customized to the nature of time series. Nonetheless, lossless data compression techniques can still be applied to the data independently after segmentation methods are applied.

Many lossy data reduction methods exist such as *Discrete Fourier Transform (DFT)* [30], *Discrete Wavelet Transform (DWT)* [17], *Piecewise Aggregate Approximation (PAA)* [15], *Singular Value Decomposition (SVD)* [31], *Symbolization* [23, 25], *Histograms* [2], data cube [22] and wave-pattern [34]. These reduction techniques focus on the global patterns of time series instead of individual data point so errors on individual data points vary widely (without bound) and unpredictably. Some recent studies have been proposed to provide probabilistic or even deterministic error bounds on individual data points [10, 11], but these algorithms have to know the whole length of the time series and work on the data offline.

Linear segmentation is another widely used lossy data reduction method for many practical applications [24, 12, 19, 29, 36] due to its simplicity. Linear segmentation methods approximate time series through *Piecewise Linear Approximation (PLA)* [16], which divides a time series into segments and uses a linear function to approximate each segment. Linear segmentation based methods can be categorized into two classes: offline segmentation and online segmentation. Offline segmentation methods, such as Top-down/Bottom-up algorithm [16] and evolutionary computation [8], need to obtain the whole time series before processing it. Online segmentation methods process the data point on the fly as each one is read. Since online segmentation methods need to process data in almost real time and continuously, they have strong requirement on the efficiency of the algorithm.

An optimal solution for the linear segmentation is proposed by Bellman [3]: given a certain number of segment k and a time series whose length is n, an optimal PLA result that minimizes the holistic approximation error (i.e., the Euclidean distance between the approximation and the original time series) is found by dynamic programming with a cost of $O(kn^2)$. To obtain this optimal result on the fly, we need continually rerun the dynamic programming algorithm when a new data point arrives, which is too expensive for the applications whose data volume is large. Consequently, greedy methods [1, 16, 28] are used in these applications.

The *Sliding Window (SW)* algorithm [1] is a classic online segmentation algorithm. It uses the first data point of a time series as the starting data point of a segment and tries to put the next data point into this segment. The straight line that connects the current data point and the starting data point is used to approximate the current segment. Every time when a new data point is read, the approximation error needs to be calculated again based on the vertical deviation between all data points and the approximation line. Once the approximation error of the current segment exceeds a given error bound, we know that the previous data point is the endpoint of the current segment. Then we take the previous data point as the new starting data point of the next segment. The above process is repeated until the end of the time series is reached. Subsequent studies have proposed some improvements to reduce the complexity of SW [18, 35]. Keogh et al. [16] present a method called SWAB which combines the SW algorithm with a Bottom-Up mechanism. Palpanas et al. [28] report a technique to reduce the complexity of SWAB and SW to linear time.

However, the goal of the above-mentioned works [1, 3, 16, 28] is to minimize the holistic approximation error (e.g., the Euclidean distance between the approximation and the original time series) given a certain amount of information (e.g., the number of segments), where the best approximation result is the one with the lowest holistic error. This goal is different from that of our work: minimizing the amount of information used to represent the time series given an error bound on each data point (i.e., the number of parameters used to represent the time series), where the best approximation result should be the one with the lowest amount of information used. Therefore, these methods do not satisfy the requirement of our problem and can not be used to solve our problem.

The most recent work with the same problem setting as ours is an online PLA segmentation method named *Feasible Space Window (FSW)*, proposed by Liu et al. [24], which introduces the concept of *Feasible space (FS)*. The feasible space is an area in the time series data value space so that any straight line in this area can approximate each data point within a given error bound. FSW aims to find the farthest segmenting point to make each segment as long

as possible given an error bound on each data point. The FSW algorithm is the state of the art for the problem addressed in this paper. Therefore, we use the FSW algorithm as the baseline in the experimental study and describe it in detail in subsection 3.2.

Lemire [21] and Fuchs et al. [9] introduce two Piecewise Polynomial Approximation (*PPA*) methods to approximate time series data by polynomial functions. The goal of the above two papers is to minimize the total approximation error given a certain number of information (specifically, the information of [21] is a given model complexity and the information of [9] is a given set of polynomials of orders: 0, 1, 2, ..., k). In contrast, the goal of our method is to minimize the amount of information given a certain error bound on each data point. Moreover, our method is more generic in terms of candidate functions, i.e, besides polynomial functions, we can also use other kinds of functions (e.g., exponential functions, etc.) as the candidate functions while the above two methods use only polynomial functions.

Joseph [27] proposes an algorithm to fit straight lines between data ranges through transferring the problem into the coefficient space. There are two differences between this work and our FCS algorithm: First, this early work only solves the problem in the case of straight lines, which is the same as what the FSW does. Fitting non-linear curves (especially, curves of high-order polynomials) into a sequence of data ranges is the major challenge addressed by our FCS algorithm. Second, the method of this work is also different from ours. Specifically, this work directly constructs the coefficient space and obtains the boundaries based on the given approximation function without any preprocessing such that its resulted coefficient space is always one dimension higher than ours and hence the computation is more complicated and the cost is higher.

3 Preliminaries

In this section, we first provide a formal definition of our problem in subsection 3.1 and then explain the most recent algorithm for solving this problem, the Feasible Space Window algorithm (FSW), in subsection 3.2. The symbols frequently used in the following sections are summarized in Table 1.

Symbol	Meaning
δ	A given error bound on each data point
P	A time series
F	A set of candidate functions
p_i	The i^{th} data point in time series
p_{start}	The starting point
p_{next}	The next coming data point
p_{t_e}	The data point at which the FCS becomes empty
n_p	The number of parameters used to represent a subsequence
n_{tp}	The number of parameters used to represent a time series
n_s	The number of segments
n_c	The number of coefficients of a function
$f_j(x)$	The j^{th} approximation function
u	An upper boundary line
l	A lower boundary line
hp	A hyperplane
hf	A hyperface
hp_m	An m-dimensional hyperplane
hf_m	An m-dimensional hyperface
hp_u	An upper boundary hyperplane
hp_l	A lower boundary hyperplane

Table 1: Frequently used symbols

3.1 Problem Statement

Given a time series $P = (p_1, p_2, \ldots, p_n)$, an error bound δ and a set of candidate functions F, our problem is to divide P into k continuous segments $S_1, S_2, \ldots, S_k$:

$$S_1 = (p_1, p_2, \ldots, p_{c_1}),$$

$$S_2 = (p_{c_1}, p_{c_1+1}, \ldots, p_{c_2}),$$

$$\ldots$$

$$S_k = (p_{c_{k-1}}, p_{c_{k-1}+1}, \ldots, p_{c_k}),$$

such that
(i) each segment S_j is approximated by a candidate function $f_j(x)$ in F with the error bound δ on each data point, more formally,

$$\tilde{p}_i = \begin{cases} f_1(i) & i = 1, \ldots, c_1, \\ f_2(i) & i = c_1, \ldots, c_2, \\ \ldots & \\ f_k(i) & i = c_{k-1}, \ldots, c_k, \end{cases} \quad (1)$$

satisfying

$$distance(\tilde{p}_i - p_i) \leq \delta;$$

and
(ii) the total number of parameters used to represent P (denoted as n_{tp}) is minimized.

Intuitively, in order to represent the approximation result of a time series, not only the values of coefficients of the approximation functions but also some other parameters, such as, the value of the starting point and the timestamp of each segmenting point, should be recorded as the approximation parameters of a time series. In this paper, in order to achieve smooth approximation, which is an important property desired in subsequent mining phases of the time series, we require the endpoint of the approximation function for the current segment to be the starting point of the approximation function for the next segment.

Thereby, the number of parameters needed to represent the first segment, which is approximated by an approximation function with n_c coefficients, is n_c+1 (i.e, n_c-1 coefficient values, a value of the starting point and a time timestamp of the first segmenting point) while that of each following segment is n_c (do not need the value of the starting point any more). Formally,

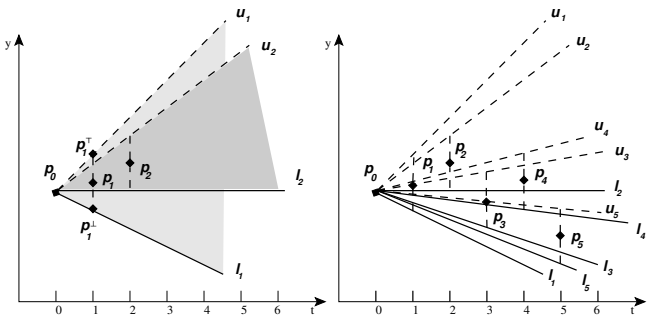
$$n_{tp} = \sum_i (n_{c_i} * n_{s_i}) + 1, \quad (2)$$

where n_{c_i} is the number of coefficients of the i^{th} candidate function and n_{s_i} is the number of segments approximated by the i^{th} candidate function.

3.2 Feasible Space Window

Liu et al. [24] propose a method to achieve the compact segmentation by PLA, which is named Feasible Space Window (FSW). The FSW algorithm can find the farthest segmenting point of each segment with the error bound guarantee on each data point through a concept called *Feasible Space* (FS). The FS is an area in the data value space of a time series such that any straight line in this area can approximate each data point of the corresponding segment within a given error bound.

Figure 3(a) shows an example of the FS. Suppose the error bound is δ, and p_0 is the starting data point of a time series which is also required to be the starting point of the approximation line (i.e., the line that approximates the data points). When we read the second



(a) Feasible space (b) Determining a segment

Figure 3: Example of FSW algorithm

data point $p_1(x_1, y_1)$, we know that the y-coordinate of the approximation line at timestamp x_1 must be between the points $p_1^{\top}$ and $p_1^{\perp}$ which are the upper and lower boundary points of p_1 and $|\overline{p_1^{\top}, p_1}| = |\overline{p_1, p_1^{\perp}}| = \delta$. Therefore, any line between the upper line u_1 and the lower line l_1 satisfies the error bound requirement for p_1, and the region (the light gray region in the figure) between these two lines is the FS after reading the data point p_1.

The FS is incrementally updated when new data points are read. For example in Figure 3(a), we read the next data point p_2 and similarly obtain two boundary lines u_2 and l_2. The area between u_2 and l_2 is the FS for p_2. We intersect this FS with the previous FS and the resultant region (the dark gray region in the figure) becomes the current FS, which is the region for any approximation line that can satisfy the error bound requirement for both p_1 and p_2.

This FS update process is repeated until the FS becomes empty at t_e. When the FS becomes empty, it means we can not approximate any more following data points (including the current data point) by a straight line within the error bound. Hence the previous data point p_{t_e-1} will be the endpoint of the current segment and also the new starting data point of the next segment.

Figure 3(b) shows the process of determining the whole segment by the FSW algorithm. After p_2, we read the next two data points p_3 and p_4 one by one and update the new FS to $[u_3, l_2]$. After we read p_5, the new FS becomes empty because the lowest upper line u_5 is below the highest lower line l_2. Therefore, the previous data point p_4 is the segmenting point and the line connecting p_0 and p_4 is the approximation result of data points between p_0 and p_4. We use p_4 as the starting data point of the next segment and repeat this FS update process until the end of the time series is reached. As we can see, the FSW algorithm provides the linear approximation as each data point is read, so it is an online algorithm.

4 Feasible Coefficient Space

Feasible space is an intriguing idea for determining the compact approximation of time series. However, it is difficult to apply the FSW algorithm to non-linear functions. The idea of FSW is to use a starting data point and a next data point to determine the boundaries of the feasible space of the approximation. Nonetheless, most non-linear functions have more than two coefficients and their approximation boundaries can not be determined by only two data points.

A method called the *FCS algorithm* is proposed to address this challenge: instead of finding the boundaries of feasible space as FSW does, we find the boundaries of the feasible values for the functions' coefficients – using two data points, we can uniquely identify such boundaries in the coefficient coordinate system. With these boundaries, we can determine a space called the *Feasible Coefficient Space (FCS)* in which each point is a feasible set of values for the coefficients of the candidate function.

Given a time series P, a certain error bound δ and a candidate function $f_j(x)$, the overview of the FCS algorithm is as follows. When the next data point p_{next} arrives, we derive two inequalities based on p_{start} (the starting data point), p_{next} and δ to determine two boundaries for the FCS of this function. Then we read the next data point to form two new boundaries and intersect them with the existent FCS to obtain a new FCS. The FCS is incrementally narrowed while the data points arrive continuously and finally becomes empty at a certain data point p_{t_e}, which means we cannot approximate any more following data points (including p_{t_e}) by the given candidate function with a given error bound on each data point. Therefore, we take the previous data point p_{t_e-1} as the segmenting point of the current segment and the starting data point of the next segment. The above process is repeated until the time series is finished.

In this paper, we focus on the FCS algorithm for a few types of commonly used functions (polynomial functions of different orders, exponential functions), although our presented method can be extended to other types of functions (e.g., logarithmic functions) straightforwardly.

4.1 Second-order Polynomials

In this subsection, we present the FCS algorithm for the second-order polynomial function (or called quadratic function). A second-order polynomial function is in the form of Equation (3) where a, b and c are coefficients of this function:

$$y = ax^2 + bx + c. \tag{3}$$

As the problem defines, the first data point $p_0(x_0, y_0)$ of the time series must be on the approximation curve. Hence we have

$$y_0 = ax_0^2 + bx_0 + c. \tag{4}$$

When a second data point $p_1(x_1, y_1)$ is encountered, if we approximate this data point by the quadratic function, the approximation value of y_1 on the curve is

$$\tilde{y}_1 = ax_1^2 + bx_1 + c. \tag{5}$$

Combining Equations (4) and (5), we have

$$\tilde{y}_1 = y_0 + a(x_1^2 - x_0^2) + b(x_1 - x_0). \tag{6}$$

Since we require the approximation error of each data point cannot exceed the user-specified error bound δ, $\tilde{y}_1$ must fall in the interval $[y_1 - \delta, y_1 + \delta]$. Therefore, we have the following inequalities:

$$y_0 + a(x_1^2 - x_0^2) + b(x_1 - x_0) \leq y_1 + \delta; \tag{7}$$

$$y_0 + a(x_1^2 - x_0^2) + b(x_1 - x_0) \geq y_1 - \delta. \tag{8}$$

Using the above inequalities, we can construct a 2-dimensional FCS in the coefficient coordinate system of the quadratic function with axes a and b (c is omitted when we combine Equations (4) and (5) and can be obtained by $c = y_0 - ax_0^2 - bx_0$ given the values of a and b.).

As shown in Figure 4, Inequalities (7) and (8) describe two parallel lines l_1 and u_1 indicating the lower and upper boundaries, between which is the current FCS, where each point is a feasible set of the coefficients' values of the quadratic function. When we read a new data point $p_2(x_2, y_2)$, we try to incorporate it in the current segment through similarly obtaining two other parallel lines l_2 and

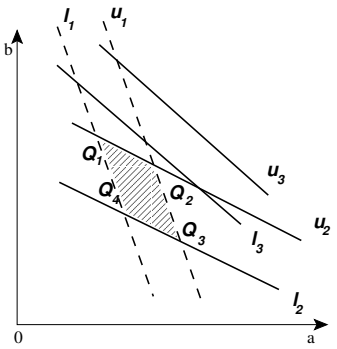


Figure 4: Feasible coefficient space for quadratic functions

u_2, between which is the FCS satisfying the error bound requirement for p_0 and p_2. If l_2 and u_2 are not parallel to the previous lines l_1 and u_1, the intersection of the previous FCS (area between l_1 and u_1) and the current FCS (area between l_2 and u_2) is the new FCS (the shaded polygon area in Figure 4).

If this new FCS is not empty, we continue to obtain the boundaries l_3 and u_3 for $p_3(x_3, y_3)$ and narrow the FCS (a polygon) by these two lines. This process is repeated until the FCS becomes empty at t_e when we can not incorporate the following data points (including p_{t_e}) in the current segment and approximate this segment by a quadratic function with a certain error bound on each data point. Therefore, we use the previous data point p_{t_e-1} as the endpoint of the current segment and also the new starting data point of the next segment and then continue the above process

Algorithm: The FCS algorithm for quadratic functions is named *FCSP2* and the pseudo-code is shown in Figure 5 where the current FCS (a convex polygon), the starting data point, the next data point and the error bound are denoted as g, p_{start}, p_{next}, and δ, respectively. When p_{next} arrives, Inequalities (7) and (8) define two lines: u and l which are the upper and lower boundaries of the FCS for p_{next}. In order to obtain the intersection of this FCS and g, for each edge of g, we calculate the intersecting points with l. Since g is convex, at most two edges of g intersect with l and g is divided into two parts by l. We remove the lower part of g and the other part is new g (*how to define a part is lower or upper will be presented in Subsection 4.2*). Similarly, we use u to divide the new calculated g and cuts the upper part. Note that g is still convex after it is cut by u and l, and the resultant polygon g' is the new FCS after processing the data point p_{next}.

Complexity Analysis: In the algorithm FCSP2, the most frequently executed operation is the calculation of intersecting points between two generated boundary lines (l and u) and the current FCS, which takes constant time. Suppose we have obtained $n+1$ data points before we get p_{next}, which means we have already generated n pairs of lines, in the worst case, these lines could make up a polygon with $2n$ edges. The number of times of computing intersection is $2n$ for either l or u and $4n$ them together. Therefore, the worst case computational cost of the FCSP2 for one particular data point is

$$C_2 = 4n \in O(n).$$

Furthermore, based on the proof in [27], the amortized complexity of the FCSP2 per data point is only $O(1)$.

4.2 M^{th}-order Polynomials

In this subsection, we present the FCS algorithm of m^{th}-order polynomials with $m \geq 3$. Firstly, we consider the case of $m = 3$.

Algorithm **FCSP2**

Input: g : the current current polygon; p_{start} : the starting data point; p_{next} : the next data point; δ : the max_error bound.
Output: g' : the remained polygon.

Construct two lines l and u by p_{start} and p_{next}
according to Inequalities (7) and (8);
If g is empty
  $g' \leftarrow$ the space between l and u;
Else
  For each edge e_i of g
    calculate the intersection points between e_i and l;
  based on the intersection information,
    cut off the part lower than l from g;
  For each edge e_j of the current g
    calculate the intersection points between e_j and u;
  based on the intersection information,
    cut off the part upper than u from g;
  $g' \leftarrow$ the remained part of g;
Return g'
End FCSP2

Figure 5: Algorithm FCSP2

The 3^{rd}-order polynomial function is also called the cubic function which is in the form of Equation (9) where a, b, c and d are coefficients of the function.

$$y = ax^3 + bx^2 + cx + d, \tag{9}$$

Similar to the case of quadratic functions, we use the starting data point $p_0(x_0, y_0)$ and the approximate value of the following data point $p_1(x_1, y_1)$ to obtain a pair of equations as follows:

$$y_0 = ax_0^3 + bx_0^2 + cx_0 + d, \tag{10}$$

$$\tilde{y}_1 = ax_1^3 + bx_1^2 + cx_1 + d. \tag{11}$$

Combining Equations (10) and (11), we have

$$\tilde{y}_1 = y_0 + a(x_1^3 - x_0^3) + b(x_1^2 - x_0^2) + c(x_1 - x_0). \tag{12}$$

As the problem defines, the approximate value $\tilde{y}_1$ should fall into the interval $[y_1 - \delta, y_1 + \delta]$ so that we have

$$y_0 + a(x_1^3 - x_0^3) + b(x_1^2 - x_0^2) + c(x_1 - x_0) \leq y_1 + \delta, \tag{13}$$

$$y_0 + a(x_1^3 - x_0^3) + b(x_1^2 - x_0^2) + c(x_1 - x_0) \geq y_1 - \delta. \tag{14}$$

In the 3^{rd}-order coefficient coordinate system with axes a, b and c, Inequalities (13) and (14) describe two bounding planes and the space between these bounding planes is a 3-dimensional FCS. Moreover, we obtain the next data point $p_2(x_2, y_2)$ to derive a new pair of bounding planes and use these planes to cut the FCS and generate a new FCS. In this case, the generated FCS is a 3-dimensional polyhedron. This process is repeated for the following data points to incrementally update the FCS until t_e when the FCS becomes empty. We segment the time series at the previous data point p_{t_e-1}, take p_{t_e-1} as the new starting point and start a new round of approximation since p_{t_e}.

It is easy to generalize the quadratic and cubic polynomials to the case of m^{th}-order polynomials, which are in the following form:

$$y = a_m x^m + a_{m-1} x^{m-1} + ... + a_1 x + a_0. \tag{15}$$

Given the starting data point and the coming data point, we can construct two $(m-1)$-dimensional hyperplanes in the m^{th}-order coefficient coordinate system as the boundaries of the m-dimensional

Algorithm **FCSPm**

Input: h : the current current (m-dimensional) hyperhedron,
p_{start} : the starting data point, p_{next} : the next data data point,
δ : the max_error bound
Output: h' : the remained m-dimensional hyperhedron

Construct two $(m-1)$-dimensional hyperplanes hp_l and hp_u
by p_{start} and p_{next} according to Inequalities (16) and (17);
If h is empty
$h' \leftarrow$ the space between hp_l and hp_u;
Else
For each $(m-1)$-dimensional hyperface of h
calculate the intersected $(m-2)$-dimensional hyperface
with hp_l;
based on the intersection information,
cut off the part lower than hp_l from h;
For each $(m-1)$-dimensional hyperface of h
calculate the intersected $(m-2)$-dimensional hyperface
with hp_u;
based on the intersection information,
cut off the part upper than hp_u from h;
$h' \leftarrow$ the remained part of h;
Return h'
End FCSPm

Figure 6: Algorithm FCSPm

FCS through the following inequalities:

$$\tilde{y}_1 = y_0 + a_m(x_1^m - x_0^m) + a_{m-1}(x_1^{m-1} - x_0^{m-1}) + ... + a_1(x_1 - x_0) \geq y_1 - \delta, \quad (16)$$

$$\tilde{y}_1 = y_0 + a_m(x_1^m - x_0^m) + a_{m-1}(x_1^{m-1} - x_0^{m-1}) + ... + a_1(x_1 - x_0) \leq y_1 + \delta. \quad (17)$$

The hyperhedron between these two boundaries is a m-dimensional FCS. When a new data point is read, we generate a new pair of $(m-1)$-dimensional hyperplanes and use them to cut the current m-dimensional hyperhedron (the current FCS) and obtain the intersecting hyperhedron as the new FCS.

Algorithm: When $m > 3$, the FCS of m^{th}-order polynomials will become a high-dimensional hyperhedron. We present an algorithm to generate this high-dimensional FCS (denoted as *FCSPm*) in Figure 6. In this algorithm, we use h, h', $hyperfaces$ and $hyperplanes$ to denote the current FCS, the new FCS, the faces of hyperhedron and high-dimensional planes generated by inequalities, respectively. The starting data point, the next data point and the error bound are defined and denoted similarly to their counterparts in Figure 5. When the new data point arrives, Inequalities (16) and (17) determine two hyperplanes denoted as hp_l and hp_u which are used to update the current FCS. The update process is similar to that of the FCSP2 algorithm.

To cut a m-dimensional FCS, we need determine a part of $(m-1)$-dimensional hyperface is the lower or upper part regarding to a $(m-1)$-dimensional hyperplane. For example, in the case of quadratic functions ($m = 2$), we need to determine the relative lower and upper parts of an edge (1-dimensional hyperface) regarding a line (1-dimensional hyperplane). Given a $(m-1)$-dimensional hyperplane in a m-dimensional coefficient coordinate system cutting the whole space into two parts, we define that a part of a $(m-1)$-dimensional hyperface is a lower (or upper) part regarding to this $(m-1)$-dimensional hyperplane if this part of the hyperface is contained in the lower (or upper) part of the space regarding the same $(m-1)$-dimensional hyperplane. The lower and upper part of the space is defined according to a selected coefficient axis called *pilot axis*: if a_m is the pilot axis, we define the upper part of the space to be the part containing $+\infty$ along the a_m axis, and the other part of this space is the lower part of space. For example, in Figure 4, choosing axis b as the pilot axis, the edge Q_1Q_4 is the upper part regarding the line l_2 because this edge is contained in the upper part of the space.

Complexity Analysis: For the case of m^{th}-order polynomials, suppose we have obtained $n+1$ data points before we get p_{next} and constructed n pairs of $(m-1)$-dimensional hyperplanes. In the worst case, these $(m-1)$-dimensional hyperplanes result in a m-dimensional hyperhedron with $2n$ $(m-1)$-dimensional hyperfaces. Each $(m-1)$-dimensional hyperface is described by $2(n-1)$ $(m-2)$-dimensional hyperfaces and, similarly, each $(m-2)$-dimensional hyperfaces can be described by $2(n-2)$ $(m-3)$-dimensional hyperfaces if $m > 3$. This process keeps on going until it reaches the 1-dimensional hyperfaces (i.e., lines).

Calculating the intersection of two $(m-1)$-dimensional hyperplanes is denoted as $Cost_{m-1}$, which takes constant time. In FCSPm, the processes of using the lower $(m-1)$-dimensional hyperplane ($hp_{l_{m-1}}$) to cut one of the $(m-1)$-dimensional hyperfaces (hf_{m-1}) of h is as follows: Firstly, we calculate the intersection of $hp_{l_{m-1}}$ and hf_{m-1} resulting a $(m-2)$-dimensional hyperplane denoted by hp_{m-2} and the cost is suppose to be constant denoted as $Cost_{m-1}$. Then, for each $(m-2)$-dimensional hyperface hf_{m-2} describing the previous $(m-1)$-dimensional hyperface hf_{m-1}, we calculate the intersection of it and hp_{m-2}, which is a $(m-3)$-dimensional hyperplane. We further use this $(m-3)$-dimensional hyperplane to cut $(m-3)$-dimensional hyperfaces describing the hf_{m-2} and get a $(m-4)$-dimensional hyperfaces. This process is executed iteratively until it reaches the calculation of the intersection of two lines and we have the total cost (denoted as C_m) as follows:

$$C_m = 2nCost_{m-1} + 2n * 2(n-1)Cost_{m-2} + \cdots + 2n * 2(n-1) * ... * 2(n-(m-2))Cost_1,$$

where $Cost_1$, $Cost_2$, ... and $Cost_{m-1}$ are different constant values. When $n >> m$, C_m is dominated by the last term, which has the worst case complexity of $O(n^{m-1})$. Although the complexity is polynomial, when m is large, the computation and implementation cost will also become very large. Therefore, we do not use polynomials with orders higher than two if it is not essential.

4.3 Exponential Functions

In this subsection, we will present the FCS algorithm of exponential functions which are in the following form in general:

$$y = be^{ax}.$$

It can be transformed into the linear approximation function using logarithmic rules. When $b > 0$, we take the natural logarithms of both sides of the equation and get

$$\ln(y) = \ln(b) + ax. \quad (18)$$

Similarly, the starting data point $p_0(x_0, y_0)$ should be on the curve such that

$$y_0 = be^{ax_0}, \quad (19)$$

namely,

$$\ln(y_0) = \ln(b) + ax_0 \ (y > 0, b > 0). \quad (20)$$

When we get another data point $p_1(x_1, y_1)$, for the approximate value of y_1, we have

$$\ln(\tilde{y}_1) = \ln(b) + ax_1 \ (\tilde{y}_1 > 0, b > 0). \quad (21)$$

Combining Equations (20) and (21), we obtain

$$\ln(\tilde{y}_1) = \ln(y_0) + a(x_1 - x_0). \quad (22)$$

We know that $\tilde{y}_1$ is within the error bound, thus $\tilde{y}_1 \in [y_1-\delta, y_1+\delta]$. Here we assume $y_1 - \delta > 0$ (if not, we iteratively multiply δ with 0.5 until it stands). Note that the natural logarithmic function is a monotonic increasing function with $(0, +\infty)$ as the definitional domain, we combine the previous two inequalities and obtain

$$\ln(y_0) + a(x_1 - x_0) \leq \ln(y_1 + \delta)\ (y_0, y_1 > 0), \quad (23)$$

$$\ln(y_0) + a(x_1 - x_0) \geq \ln(y_1 - \delta)\ (y_0, y_1 - \delta > 0). \quad (24)$$

Then we have

$$\frac{\ln(y_1 - \delta) - \ln(y_0)}{x_1 - x_0} \leq a \leq \frac{\ln(y_1 + \delta) - \ln(y_0)}{x_1 - x_0}. \quad (25)$$

This inequality defines a pair of boundary points for the coefficient a. Every time when a new data point comes, we obtain a new pair of such boundary points and incrementally update the feasible value range (here is 1-dimensional FCS) for coefficient a until the FCS becomes empty. Actually, this is just the linear segmenting problem. The difference is that in the linear segmenting case, Inequality (25) becomes

$$\frac{y_1 - y_0 - \delta}{x_1 - x_0} \leq a \leq \frac{y_1 - y_0 + \delta}{x_1 - x_0}. \quad (26)$$

Therefore, the FCS algorithm used for the exponential function is the same as the linear function. Furthermore, the aforementioned FCS algorithm FCSPm is applicable for both the linear function and the exponential function because the linear function is the 1^{st}-order polynomial whose high-dimensional coefficients are zero.

Complexity Analysis: In the complexity aspect, the exponential function is also similar to the linear function. No matter how many data points have been processed, we only need to maintain two points to indicate the current feasible value range of coefficient a (1-dimensional FCS). Therefore, when the next data point arrives, we only need to compare the values of two pairs of boundary points to obtain the new FCS such that the computation complexity of exponential function is $C_e = C_1 = O(1)$

5 Adaptive Approximation Algorithm

The objective of our method is to minimize the number of parameters used to represent the time series given an error bound on each data point. Suppose there is a given starting data point p_{start} and an appropriate FCS algorithm for each candidate function. The overview of the AA algorithm is as follows: we read the data points one by one and approximate these points through the respective FCS algorithms of the candidate functions. For each candidate function, the corresponding FCS is incrementally narrowed while the data points arrive continuously and turns into empty at t_e, so we take p_{t_e-1} as the segmenting point of the current segment and the new starting data point of the next segment, and then we continue to read and approximate the following data points. This process is repeated by all the candidate functions until encountering the data point at which each candidate function has encountered at least one segmenting point. At this point, for each candidate function, we calculate the number of parameters used to represent this subsequence (denoted as n_p), then compare and choose the function with the smallest n_p as the approximation function of the subsequence. We repeat the above process until the time series is finished.

Similar with Equation 2, the number of parameters used to represent a subsequence approximated by a candidate function $f_j(x)$ can be obtained by $n_p = n_{c_j} * n_{s_j}$, where n_{c_j} is the number of coefficients of $f_j(x)$ and n_{s_j} is the number of segments generated and approximated by $f_j(x)$. However, when we calculate n_p at the data point p_t where each candidate function has encountered at least one segmenting point, we know p_{t-1} must be the segmenting point for some candidate functions but it may not be the segmenting point for other candidate functions such that these functions can approximate more data points without increasing n_p. To keep fair comparison, we heuristically tune the calculation of n_p for these functions to be $n_p = n_c * (n_s - 1) + n_c * k$ where k ranges from 0 to 1.

In this paper, we define the segments obtained by each candidate function as the *local segments* of the corresponding function and the segments obtained by the final chosen approximation function as the *global segments* of the corresponding subsequence. Thereby, for each data point, it belongs to many different local segments (generated by different candidate functions) but only one global segment (generated by the chosen approximation function). For each candidate function, if it's chosen as the approximation function of a subsequence, the segments generated by this function will be the global segments of this subsequence. Otherwise they are just local segments.

The algorithm complexity of our method basically depends on the candidate function with the highest complexity. For example, if we choose the quadratic function, the linear function and the exponential function as the candidate functions, the complexity is dominated by the quadratic function such that the worst case complexity and the amortized complexity of the AA algorithm will be $O(n)$ and $O(1)$, respectively. We provide the pseudo-code of the AA algorithm in Figure 7 and present the AA algorithm specifically in the following subsections.

5.1 Initialization

In our algorithm, we use the lists to store the parameters used to represent the time series: the approximation parameters of the global segments of the time series and those of the local segments generated by candidate functions are stored in the lists l_g and l_{l_j}, respectively, where j indicates this list is used to store the approximation parameters of the local segments generated by the j^{th} candidate function. At the beginning of the AA algorithm, we initialize the AA algorithm through using the first data point p_0 of the time series P to be the first data point of the first global segment of P (denoted as p_{first}) and also the starting data point of the first local segment of each candidate function (denoted as p_{start_j}). Furthermore, we set all lists and the feasible coefficient spaces of all candidate functions (denoted as FCS_j) to be empty and also set two flags F_{rv} and F_{cf} to be false. Here, F_{cf} indicates whether or not there has been a candidate function chosen as the approximation function and F_{rv} indicates whether the chosen approximation function is not longer feasible for further approximation.

5.2 Finding Segmenting Point

When the next data point p_{next} comes, we firstly need to check the value of flag F_{cf}. If F_{cf} is false, it means there is not a chosen approximation function so we iteratively update the FCS of each candidate function to choose an approximation function. Otherwise, we simply use the chosen function to approximate the coming data point and only update its FCS. Therefore, the process of finding segmenting data point can be divided into two cases as the followings:

Case One: If F_{cf} is false, for each candidate function $f_j(x)$ in F, we firstly update its feasible coefficient space FCS_j through the corresponding FCS algorithm $FCSA_j$. Then we check whether

the AA algorithm

Input: $P = (p_0, p_1, ...p_n, ...)$, δ : max_error bound.
$F = \{f_1(x), f_2(x), ... f_j(x), ... f_m(x)\}$
Output: l_g : list of global segmenting points
Initial: $p_{first} = p_{start1} = p_{start2} = ... = p_{startm} = p_0$
$l_g = l_{l_1} = l_{l_2} = ... = l_{l_m} = \emptyset$
$FCS_1 = FCS_2 = ... = FCS_m = \emptyset$
$F_{rv} = F_{cf} = false$

While P not finished
fetch the next data point p_{next} from P;
If $F_{cf} = false$ // have not chosen approximation function
For each candidate function $f_j(x)$ in F
$FCSA_j\ (f_j(x), FCS_j, p_{start_j}, p_{next})$;
If $FCS_j = \emptyset$
append new local segment to l_{l_j};
$p_{start_j} \leftarrow p_{next-1}$;
$FCSA_j\ (f_j(x), FCS_j, p_{start_j}, p_{next})$;
If all $l_{l_j} \neq \emptyset$
For each $f_j(x)$ in l_f
calculate n_p for points between p_{first} and p_{next-1};
choose function with the smallest n_p as $f_a(x)$;
If $p_{start_a} \neq p_{next-1}$
$F_{cf} \leftarrow true$;
Else
$F_{rv} \leftarrow true$;
append l_{l_a} to l_g;
Else //have chosen approximation function
$FCSA_a\ (f_a(x), FCS_a, p_{start_a}, p_{next})$;
If $FCS_a = \emptyset$
append new local segment to l_{l_a};
append l_{l_a} to l_g;
$F_{cf} \leftarrow false$;
$F_{rv} \leftarrow true$;

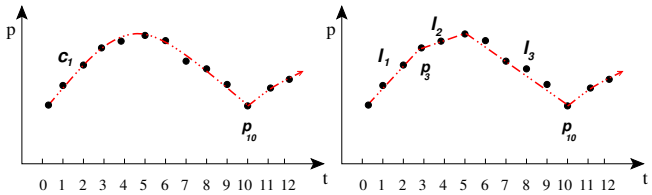
If $F_{rv} = true$ //re-initialization
$p_{first} \leftarrow p_{next-1}$;
For j from 1 to m
$l_{l_j} \leftarrow \emptyset$;
$p_{start_j} \leftarrow p_{next-1}$;
$FCSA_j\ (f_j(x), FCS_j, p_{start_j}, p_{next})$;
$F_{rv} \leftarrow false$;
Return l_g;
End the AA algorithm

Figure 7: The AA algorithm

the FCS becomes empty after update. If the FCS is empty, it means the previous data point p_{next-1} is the segmenting point of the candidate function $f_j(x)$. Therefore, for this candidate function, we save the approximation parameters of this generated local segment into the local segment list l_{l_j}, change the starting data point of this function p_{start_j} to be the previous data point p_{next-1} and then reconstruct the FCS for p_{next} based on the new starting data point through re-invoking the corresponding FCS algorithm $FCSA_j$.

When we finish the FCS update of each candidate function, we further check whether or not all local segment lists are non-empty. If so, it means each candidate function meets at least one segmenting point and we need to choose an approximation function for the subsequence ranging from p_{first} to p_{next-1}. we calculate the number of parameters used by each candidate function to represent the subsequence (denoted as n_p) and choose the function with the smallest n_p as the approximation function.

After that, we check whether the previous data point p_{next-1} is the segmenting point of $f_a(x)$. If not, we will use this function to approximate more subsequent data points until its FCS becomes empty so we change the value of F_{cf} to be true. Otherwise, we directly append the approximation parameters in its local segment



(a) Quadratic functions (b) Linear functions

Figure 8: Running example of the AA algorithm

list l_{l_a} to the global segment list l_g and change F_{rv} to be true to do re-initialization.

Case Two: If F_{cf} is true, instead of the executing processes in case one, we only update the FCS of the chosen approximation function for the coming data point p_{next} and check whether its FCS becomes empty after update. If not, we finish the process of this data point and continue to process the next data point. Otherwise, it indicates this approximation function arrives a new segmenting point p_{next-1}. Therefore, we add the approximation parameters of this new generated segment to the local segment list of this function (denoted as l_{l_a}) and append the information in this list to the global segment list l_g. Finally, we set F_{cf} to be false and F_{rv} to be true, respectively.

5.3 Re-initialization

If the value of F_{rv} becomes true after the processes in subsection 5.2, it means the re-initialization step needs to be invoked. The processes of re-initialization is as follows: (i) we firstly use the previous data point p_{next-1} to be the first data point of the following global segment p_{first} and also the starting data point of the following local segment for each candidate function; (ii) for each candidate function, we set the corresponding local segment list to be empty and re-construct the FCS for p_{next} based on the new starting data point.

We repeat the previous processes in Subsections 5.2 and 5.3 until the time series is finished. Given the same approximate error bound, the AA algorithm can achieve better extent of compactness through always finding the most compact candidate function, which owns the the smallest n_p, as the approximation function. As we can see, the approximation process of the AA algorithm only relies on the coming data point. Therefore, it is an online segmentation algorithm.

5.4 A Running Example

In this subsection, we use a running example to illustrate the AA algorithm. Figure 8 depicts the cases of approximating a time series by two candidate functions: the quadratic function and the linear function. Starting from the data point p_0, both two functions incrementally read and approximate the coming data points one by one through their FCS algorithms. When p_4 comes, the FCS of the linear function becomes empty so p_3 is the segmenting point of the linear function. However, because the FCS of the quadratic function is not empty at p_4, for the linear function, we save p_3 in its local segment list, change its starting data point to be p_3, re-construct its FCS for p_4 based on new starting data point and continue to approximate the next data point p_5. The approximation process is repeated by both two functions until p_{11} where the FCS of the quadratic function becomes empty such that both two functions encounter at least one segmenting point. Therefore, we calculate the n_p (the number of parameters used to represent the subsequence) of both functions and obtain $n_p = 3$ for the quadratic function and $n_p = 6$ for the linear function. The quadratic function is chosen

as the approximation function for the subsequence between p_0 and p_{10} because of the smaller value in n_p. For both candidate functions, we take p_{10} as the new starting data point and continue the same process to approximate the subsequent data points.

6 Experimental Study

In this section, we evaluate the performance of our AA algorithm, comparing with the state-of-the-art algorithm FSW [24]. We use the linear function, the quadratic function and the exponential function as the candidate functions in our implementation.

Our goal is to obtain a more compact approximation with an user-specified error bound on each data point. Therefore, we measure the performance by the *Number of Approximated Points per Parameter (NAPP)*, which indicates the compactness of the approximation, NAPP is defined as the number of parameters used to represent a time series (generated by Equation 2) divided by the number of data points of this time series. Even though the approximation errors are bounded by a given threshold, we also measure the actual average approximation error, which indicates how accurate the approximation actually are. The average approximation error is defined as the sum of individual approximation errors on all the data points divided by the number of the data points.

The error bounds are expressed as relative values in comparison to the maximum value of the data points of the time series.

6.1 Synthetic Datasets

In this subsection, we evaluate the algorithms using four synthetic time series datasets: a linear time series dataset, a quadratic time series dataset, a exponential time series dataset and a mixed time series dataset. Each dataset is composed of 1000 data points which are obtained by sampling 40 data points from each of 25 randomly valued functions. For the linear, quadratic and exponential time series datasets, the functions are linear, quadratic and exponential synthetic functions, respectively. For the mix time series dataset, each function is randomly choose among the aforementioned three kinds of synthetic functions. The definitions of these synthetic functions are as follow:

(1) Linear Function

$$f(x) = ax + b + \epsilon,$$

where the values of coefficients a and b are randomly chosen from the ranges of $[-1, 1]$ and $[-10, 10]$, respectively.

(2) Quadratic Function

$$f(x) = ax^2 + bx + c + \epsilon,$$

where a is randomly chosen from -1 or $+1$, b and c are randomly chosen from the ranges of $[-10, 10]$ and $[-20, 20]$, respectively.

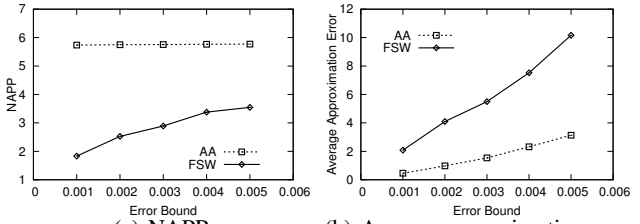
(3) Exponential Function

$$f(x) = be^{ax} + \epsilon,$$

where a is randomly chosen from the range: $[0.05, 0.1]$, and b is randomly chosen from the ranges of $[-1, -2]$ and $[1, 2]$.

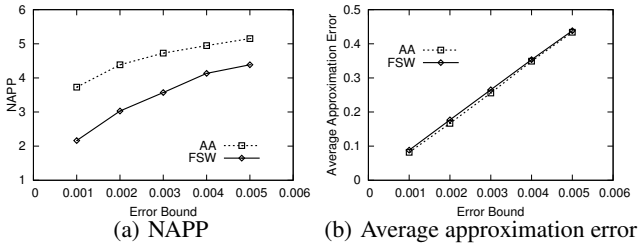
ϵ in these functions is a random value used as a random noise.

In order to evaluate the effect of the error bound on the NAPP and the average approximation error for a given time series, we choose five different values to be the relative error bound: 0.001, 0.002, 0.003, 0.004 and 0.005 and obtain the NAPP and the average approximation error of both the AA algorithm and the FSW algorithms for each error bound. The experimental results are shown in Figures 9, 10, 11 and 12, respectively.



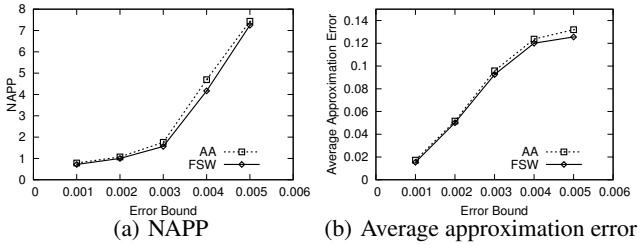
(a) NAPP (b) Average approximation error

Figure 9: Quadratic time series



(a) NAPP (b) Average approximation error

Figure 10: Exponential time series



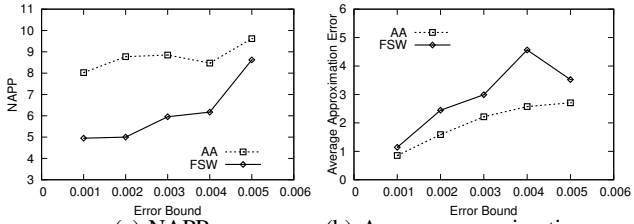
(a) NAPP (b) Average approximation error

Figure 11: Linear time series

Quadratic Time Series: Figure 9 shows the experimental result of the quadratic time series dataset. We observe that the AA algorithm outperforms the FSW algorithm in terms of both the NAPP and the average approximation error for all error bound cases. In terms of NAPP, the improvement factor of the AA algorithm over the FSW algorithm is up to 2 times when the error bound is 0.001, while the approximation error produced by the AA algorithm is only 10% of that produced by the FSW algorithm. The reason is that, for the dataset with quadratic patterns, the AA algorithm can accommodate these patterns and approximate the time series through adaptively choosing quadratic functions while the FSW algorithm can only approximate the time series by linear functions. Furthermore, when the error bound becomes larger, the average approximation error of the FSW algorithm increase dramatically with a relatively slow rise in the NAPP. In comparison, the AA algorithm keeps steady in the aspects of both NAPP and the approximation error because it has already accommodated to the patterns of the time series very well since the small error bound. This shows the robust performance of the AA algorithm in comparison to the FSW algorithm.

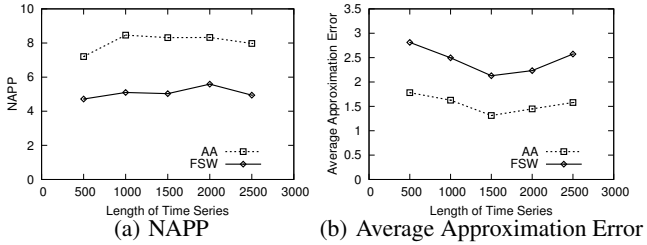
Exponential Time Series: Figure 10 shows the experimental result of the exponential time series dataset. The comparative NAPP performance of the AA algorithm and the FSW algorithm is similar to that of the quadratic time series datasets. The NAPP values of the AA algorithm are much larger than those of the FSW algorithm for all error bounds with similar average approximation errors.

Linear Time Series: The time series with linear patterns is the best case for the FSW algorithm. From Figure 11, we observe that the AA algorithm is as good as the FSW algorithm in terms of both



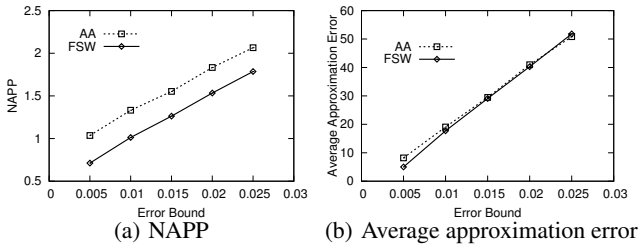
(a) NAPP (b) Average approximation error

Figure 12: Mixed time series



(a) NAPP (b) Average Approximation Error

Figure 13: Scalability, mix time series



(a) NAPP (b) Average approximation error

Figure 14: Speech time series

the NAPP and the average approximation error for all error bounds. That is because the AA algorithm adapts to linear functions to approximate the linear time series such that its approximation results are the same as those of the FSW algorithm.

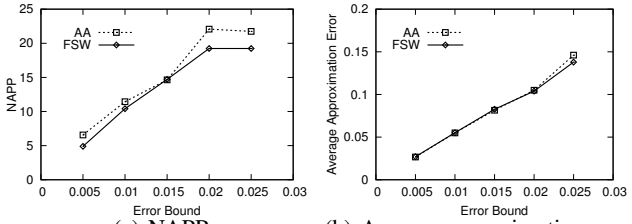
Mixed Time Series: In many real world situations, the patterns of time series data do not follow a constant rule. Hence, we use the mixed time series dataset to simulate this case and evaluate the adaptive mechanism of the AA algorithm in this situation. As shown in Figure 12, we observe that the NAPP values of the AA algorithm are 50% larger than those of FSW algorithm in most cases while the average approximation errors are much smaller than those of FSW algorithm. The result shows that the AA algorithm can adopt to the change of patterns better than the FSW does through adaptively choosing an appropriate approximation function.

Scalability: We vary the lengths of the time series from 500 to 2500 data points and set the relative error bound to 0.002. The experimental result of the mix time series is shown in Figure 13. We find that, for both the AA algorithm and the FSW algorithm, the two evaluation measures change little when the length of the time series increases. This shows that the AA algorithm scales well with the length of the time series. Results of other synthetic time series datasets show similar behavior and hence are omitted.

6.2 Real Datasets

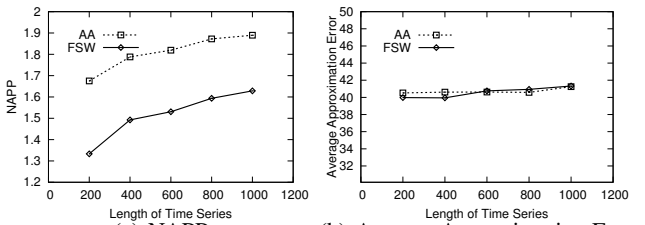
In this subsection, we evaluate the algorithms using two real world time series datasets: the Memory dataset and the Speech dataset, which have irregular patterns (e.g., neither polynomial nor exponential patterns), from UCR Time Series Data Mining Archive [14].

In order to evaluate the effect of the error bound on both the



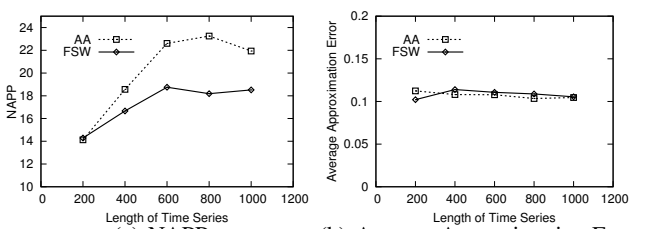
(a) NAPP (b) Average approximation error

Figure 15: Memory time series



(a) NAPP (b) Average Approximation Error

Figure 16: Scalability, speech time series



(a) NAPP (b) Average Approximation Error

Figure 17: Scalability, memory time series

NAPP and the average approximation error, we set the length of the time series to be 500 data points and vary the error bounds from 0.005 to 0.025. Then we plot the experimental results of the Speech and Memory time series datasets in Figures 14 and 15, respectively.

Speech and Memory Time Series: From these two figures, we can observe that, for both the speech and the memory datasets, the AA algorithm obtains higher NAPP values than those of the FSW algorithm with similar average approximation errors. Again, this is because the AA algorithm can adapt to different patterns using different candidate functions. This finding asserts that the AA algorithm can achieve more compact approximation than the FSW algorithm in practical workloads, even if the time series do not suit the candidate functions perfectly. The approximation error of the AA algorithm is close to that of FSW, because the patterns of these real datasets are irregular and no candidate function in the AA algorithm can perfectly accommodate these patterns.

Scalability: Because of the limited sizes of real datasets, we vary the lengths of the time series from 200 to 1000 data points and set the relative error bound to 0.02. The experimental results are plotted in Figure 16 and 17. An inspection of these figures reveals: Firstly, for both the AA algorithm and the FSW algorithm, when the length of the time series is relatively small (ranges from 200 to 600 data points), an increase of the length yields a rise of the NAPP, and when the length of the time series is relatively large (ranges from 600 to 1000 data points), the variations of the NAPP are small with an increase of the time series length. Second, the changes of the average approximation errors of both two methods are always marginal when the length of the time series increases. Overall, for real datasets, the AA algorithm also scales well with

the length of the the time series.

6.3 Summary

Experiments in this section validate that the AA algorithm achieves more compact representation of time series than the FSW algorithm does for almost all the time series datasets except the linear case. When the time series follows linear patterns, the performance of the AA algorithm is the same as the FSW algorithm. At the same time, the actual average error produced by the AA algorithm is smaller than that produced by the FSW algorithm in some cases.

7 Conclusions

We proposed an online segmentation algorithm which approximates time series by a set of candidate functions (polynomials of different orders, exponential functions, etc.) and adaptively chooses the most compact one as the pattern of the time series changes. We further proposed a novel method to efficiently generate the compact approximation of a time series in an online fashion for several types of candidate functions. This method incrementally narrows the feasible coefficient space of candidate functions in the coefficient coordinate system and find the farthest data point a segment can approximate given an error bound on each data point. Extensive experimental results show that our algorithm outperforms the state-of-the-art algorithm: FSW, in terms of the NAPP. Moreover, in some cases, our algorithm results in much lower actual errors than those caused by the FSW algorithm for the same error bound.

In future research, we will investigate how to derive the feasible coefficient space for more types of candidate functions, such as trigonometric functions and logarithmic functions, to meet more approximation needs.

Acknowledgment

This work is supported by the Australian Research Council's Discovery funding scheme (project numbers DP0880250 and DP08 80215).

8 References

[1] U. Appela and A. V. Brandta. Adaptive sequential segmentation of piecewise stationary time series. In *Information Science*, pages 27–56, 1983.

[2] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *PODS*, pages 1–16, 2002.

[3] R. Bellman. On the approximation of curves by line segments using dynamic programming. In *Communications of the ACM*, page 284, 1961.

[4] C. Cortes, K. Fisher, D. Pregibon, A. Rogers, and F. Smith. Hancock: A language for extracting signatures from data streams. In *SIGKDD*, pages 9–17, 2000.

[5] M. Dacorogna, R. Gencay, U. A. Muller, O. V. Pictet, and R. Olsen. *An Introduction to High-Frequency Finance*. Academic Press, 2001.

[6] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos. Compressing historical information in sensor networks. In *SIGMOD*, pages 527–538, 2004.

[7] K. Fisher and R. Gruber. Pads: Processing arbitrary data streams. In *Workshop of AT&T Labs*, June 2003.

[8] A.-C. Fu, F.-L. Chung, V. Ng, and R. Luk. Evolutionary segmentation of financial time series into subsequences. In *Evolutionary Computation*, pages 426–430, 2001.

[9] E. Fuchs, T. Gruber, J. Nitschke, and B. Sick. Online segmentation of time series based on polynomial least-squares approximations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(12):2232–2245, 2010.

[10] M. Garofalakis and P. B. Gibbons. Wavelet synopses with error guarantees. In *SIGMOD*, pages 476–487, 2002.

[11] M. Garofalakis and A. Kumar. Deterministic wavelet thresholding for maximum-error metrics. In *PODS*, pages 166–176, 2004.

[12] X. Ge and P. Smyth. Segmental semi-markov models for endpoint detection in plasma etching. In *TOSE*, 2001.

[13] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss. Quicksand: Quick summary and analysis of network data. Technical Report 2001-43, DIMACS, 2001.

[14] E. Keogh and T. Folias. The UCR time series data mining archive. In *http://www.cs.ucr.edu/ eamonn/TSDMA*, 2002.

[15] E. J. Keogh, K. Chakrabarti, M. J. Pazzani, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *Knowl. Inf. Syst.*, 3(3):263–286, 2001.

[16] E. J. Keogh, S. Chu, D. Hart, and M. J. Pazzani. An online algorithm for segmenting time series. In *ICDM*, pages 289–296, 2001.

[17] F. Kin-Pong Chan, A. Wai-chee Fu, and C. Yu. Haar wavelets for efficient similarity search of time-series: With and without time warping. *TKDE*, 15(3):686–705, 2003.

[18] A. Koski, M. Juhola, and M. Meriste. Syntactic recognition of ecg signals by attributed finite automata. *Pattern Recognition*, 28(12):1927–1940, 1995.

[19] V. Lavrenko, M. Schmill, D. Lawrie, P. Ogilvie, D. Jensen, and J. Allan. Mining of concurrent text and time series. In *SIGKDD*, pages 37–44, 2000.

[20] D. A. Lelewer and D. S. Hirschberg. Data compression. *ACM Comput. Surv.*, 19(3):261–296, 1987.

[21] D. Lemire. A better alternative to piecewise linear time series segmentation. In *SIAM Data Mining*, pages 545–550, 2007.

[22] X. Li and J. Han. Mining approximate top-k subspace anomalies in multi-dimensional time-series data. In *VLDB*, pages 447–458, 2007.

[23] J. Lin, E. J. Keogh, S. Lonardi, and B. Y. chi Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *DMKD*, pages 2–11, 2003.

[24] X. Liu, Z. Lin, and H. Wang. Novel online methods for time series segmentation. *TKDE*, 20(12):1616–1626, 2008.

[25] V. Megalooikonomou, Q. Wang, G. Li, and C. Faloutsos. A multiresolution symbolic representation of time series. In *ICDE*, pages 668–679, 2005.

[26] W. O'Neil. *How to Make Money in Stocks (4 edition)*. McGraw-Hill, 2009.

[27] J. O'Rourke. An on-line algorithm for fitting straight lines between data ranges. *Commun. ACM*, 24:574–578, 1981.

[28] T. Palpanas, M. Vlachos, E. J. Keogh, and D. Gunopulos. Streaming time series summarization using user-defined amnesic functions. *TKDE*, 20(7):992–1006, 2008.

[29] S. Park, S. wook Kim, and W. W. Chu. Segment-based approach for subsequence searches in sequence databases. In *ACM Symposium on Applied Computing*, pages 248–252, 2001.

[30] D. Rafiei and A. O. Mendelzon. Efficient retrieval of similar time sequences using dft. In *FODO*, pages 249–257, 1998.

[31] K. V. Ravi Kanth, D. Agrawal, and A. Singh. Dimensionality reduction for similarity searching in dynamic databases. *SIGMOD*, 27(2):166–176, 1998.

[32] H. Shatkay. Approximate queries and representations for large data sequences. In *ICDE*, pages 536–545, 1996.

[33] M. Sullivan and A. Heybey. Tribeca: A system for managing large databases of network traffic. In *USENIX Technical Conference*, 1998.

[34] L.-a. Tang, B. Cui, H. Li, G. Miao, D. Yang, and X. Zhou. Effective variation management for pseudo periodical streams. In *SIGMOD*, pages 257–268, 2007.

[35] H. J. L. M. Vullings, M. H. G. Verhaegen, and H. B. Verbruggen. Ecg segmentation using time-warping. *Advances in Intelligent Data Analysis Reasoning about Data*, 1280:275–285, 1997.

[36] C. Wang and X. S. Wang. Supporting content-based searches on time series via approximation. In *SSDM*, pages 69–81, 2000.

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