Novel Online Methods for Time Series Segmentation

Xiaoyan Liu, *Member*, *IEEE Computer Society*, Zhenjiang Lin, and Huaiqing Wang

Abstract—To efficiently and effectively mine massive amounts of data in the time series, approximate representation of the data is one of the most commonly used strategies. Piecewise Linear Approximation is such an approach, which represents a time series by dividing it into segments and approximating each segment with a straight line. In this paper, we first propose a new segmentation criterion that improves computing efficiency. Based on this criterion, two novel online piecewise linear segmentation methods are developed, the *feasible space window method* and the *stepwise feasible space window method*. The former usually produces much fewer segments and is faster and more reliable in the runtime than other methods. The latter can reduce the representation error with fewer segments. It achieves the best overall performance on the segmentation results compared with other methods. Extensive experiments on a variety of real-world time series have been conducted to demonstrate the advantages of our methods.

Index Terms—Data mining, time series, segmentation, algorithms.

1 Introduction

Time series have increasing importance in many application domains such as finance [19], medicine [18], and manufacturing [7]. Just as its name implies, a time series is a sequence of real numbers with time stamps. A typical example is the daily closing prices of stocks. In recent years, there has been a growing interest in using data mining techniques to extract useful patterns from time series databases.

However, with the development of automatic data collection and storage technologies, we are challenged to deal with large amounts of data far beyond our cognition. For example, there are about 50,000 securities trading in the US, and as many as 100,000 quotes and trades are generated every second [8]. Moreover, in telecommunication, the AT&T long-distance data stream consists of approximately 300 million records per day from 100 million customers [8]. Mining in such a huge volume of time series data can be very time consuming. In the real world, detailed analysis of each and every time point is often neither practical nor necessary. In addition, maintaining such huge databases involves a considerable expense. Therefore, reducing the data dimension becomes crucial to efficient knowledge discovery. Approximate representation of the time series is one of the solutions, and several high-level representations have been proposed, including Discrete Fourier Transform [1], [23], Discrete Wavelet Transform [4], Singular Value Decomposition [12], Piecewise Aggregate Approximation [17], [26], and Piecewise Linear Approximation (PLA) [15], [18]. Recently, a generalized dimension-reduction framework for recent-biased time series analysis has been proposed in [27]. Of these representations, PLA is one that is widely used because of its simplicity.

To represent a time series approximately, one approach is to divide the data sequence into a few segments, and the data in each segment are then approximated by a simple function, namely, a generating mechanism. This is called a segmentation problem. The outcome of segmentation can be further used to support indexing, clustering, and classification tasks in time series mining. This problem has two basic requirements: representation quality and computing efficiency. To produce the best segmentation scheme to meet the first requirement, the following criteria are used in [15]:

1. minimizing the representation error due to approximation by given K segments,
2. minimizing the number of segments such that the representation error of any segment is less than *max_error*, and
3. minimizing the number of segments such that the total representation error of all segments does not exceed *max_total_error*,

where K, *max_error*, and *max_total_error* are predefined parameters by users. Therefore, the first requirement means that a segmentation algorithm should approximate the time series with a lower representation error using as few segments as possible. In some of the literature, the segmentation scheme is only evaluated from the aspect of representation error [15], [21], [26]. That is, the best segmentation scheme usually means the one with the lowest representation error. In this paper, we take the number of segments into account as well. Thus, for a segmentation scheme whose representation error is within the tolerance of the user, the smaller the number of segments, the better the representation. The second requirement of the segmentation problem is that a segmentation algorithm should be fast enough to fit for an online real-time working environment, since most time series are collected in large amounts

- *X. Liu and H. Wang are with the Department of Information Systems, City University of Hong Kong, Kowloon, Hong Kong. E-mail: {xiaoliu, iswang}@cityu.edu.hk.*
- *Z. Lin is with the Department of Computer Science and Engineering, Chinese University of Hong Kong, New Teritorior, Hong Kong. E-mail: zjlin@cse.cuhk.edu.hk.*

Manuscript received 21 Apr. 2006; revised 1 Oct. 2006; accepted 17 Jan. 2008; published online 25 Jan. 2008.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0187-0406.
Digital Object Identifier no. 10.1109/TKDE.2008.29.

dynamically. In this paper, we propose an incremental data processing technique to improve computing efficiency.

In the literature, the segmentation methods can be categorized into two classes: online and offline. The offline methods segment the whole data sequence, and the online methods produce segments based on the data seen so far [15]. The representation error of offline methods is usually lower than that of the online methods since offline methods take a global view of the time series. We can see that both criterias 1 and 2 in the first requirement are suitable for the offline methods since these methods need the whole time series in order to determine the parameters K and *max_total_error*, and criterion 2 is suitable for the online methods since it gives the error bound on a single segment only.

Linear segmentation, in general, involves two types of approximation: linear interpolation and linear regression. The former uses the straight line connecting the two endpoints of one segment to represent the data points in the segment and generates continuous piecewise lines, while the latter uses the regression line to approximate a segment and produces a set of disjointed lines. Due to the advantages of smooth approximation and low computational complexity [15], linear interpolation became our technical choice for online segmentation algorithms.

In this paper, we focus on the online PLA and propose two novel online segmentation algorithms, which are designed according to the aforementioned two requirements, i.e., good representation and high computing efficiency, for dealing with an online data sequence. The major contributions of this paper are given as follows: We introduce a new segmentation criterion called *feasible space*, which successfully reduces the computational complexity of the classic sliding window (SW) method. Based on this criterion, a novel SW method called the Feasible Space Window (FSW) method is proposed, which always finds the farthest endpoint of a segment along the incoming data stream. In contrast with other methods, FSW dramatically reduces the number of segments given the same maximum error tolerance. We also introduce the *stepwise strategy* to the FSW method and call it the stepwise FSW (SFSW) method. The SFSW method reduces the representation error of the FSW and overcomes the deficiency of SW methods—that the new incoming data points have no influence on the finished segment—by involving a backward segmenting process. Our extensive numerical experiments show that in all test cases, both proposed methods produce far fewer segments than the classic SW method and the SW And Bottom-up (SWAB) method. They also outperform the classic SW method in representation quality. However, the SFSW method produces a lower representation error than the FSW method with almost the same number of segments. Our numeric experiments indicate that among these methods, the FSW method is the fastest and the most reliable method from the aspect of runtime.

The remainder of the paper is organized as follows: Section 2 briefly introduces related work on the existing segmentation methods. In Section 3, the new segmentation criterion is proposed, and the two novel methods, FSW and SFSW, are presented in detail. The computational complexities are analyzed in Section 4. Extensive experiments are performed to evaluate our proposed methods in Section 5. In the last section, we conclude our paper and discuss future work.

2 Related Work

Segmentation algorithms are usually used to obtain segmenting points in places where the behavior of the sequence changes significantly, and between two segmenting points, the data can be approximated by a simple function with acceptable errors. Since we are mainly concerned with the PLA segmentation approach in this paper, our segmentation problem can be stated in the following form: for a given time series $T = (a_1, a_2, \ldots, a_n)$ of length n, the goal is to divide T into a sequence of segments $S_1 S_2 \ldots S_k$ $(1 \leq k \leq n-1)$ and represent each segment with a straight line.

Shatkay and Zdonik [24], Keogh et al. [15], and Gionis and Mannila [8] have given detailed reviews of segmentation methods in their reports. In general, segmentation algorithms for time series can be categorized into two classes: offline methods and online methods. Offline methods are applied to complete sequences, and many techniques can be adopted. Dynamic programming is used in [3] and [5] to determine the total number of intervals and their location, as well as the order of the model within each segment. Top-down algorithms [15], [24] start with an unsegmented sequence and introduce one cutting point at a time, repeating this process until some stopping criterion is met. Bottom-up algorithms [13], [16] start with $n-1$ segments—each point is a segmenting point. Then, two consecutive segments are greedily merged into one according to some criteria. It is easy to see that the Top-down algorithm and Bottom-up algorithm are complementary to each other. However, the results of the Bottom-up algorithm are reported to be better than those of the Top-down algorithm in most cases [15]. Other algorithms include approximate dynamic programming [9], divide and segment [8], and evolutionary computation [6].

Online methods segment a time series while data is being gathered based on the data seen so far. The classic SW [2] algorithm is such a method. It works by initializing the first data point of a time series as the left endpoint of a segment and then trying to put one more data point into the segment in each step. The interpolating line or regression line between the two endpoints of the segment is used as the approximation, and the corresponding representation error is calculated. If the representation error of the current segment exceeds the predefined *max_error* after adding point i into it, point $i-1$ will be set as the right endpoint of the current segment. Then, point $i-1$ or point i is taken as the left endpoint of the next segment in the interpolation approximation or regression approximation, respectively. By repeating the above process, the whole time series can be piecewisely approximated by straight lines.

Some modifications have been made to the classic SW method, such as incrementing the step length by k instead of one [18] or using an adaptive step length [25]. A monotonic segmentation method is adopted in [22]. The data in the produced segment is always monotonically increased or decreased. However, this method will generate too many segments if the sequence is fluctuated frequently or full of

Algorithm 3.1 Sliding Window Segmentation Algorithm (SW)

Input: time sequence $(a_1, a_2, \cdots, a_n, \cdots)$, max_error threshold δ

Output: segmenting points $(s_1, s_2, \cdots, s_k, \cdots)$

Initial: i=1, *index*=1, $s_1 = a_1$

While not finished segmenting time series
  Index = *index* + 1; //segmenting points index
  k=i+1;
  While *Calerror*(S$(a_i, \cdots, a_k)) \leq \delta$
    k=k+1;
  End
  $s_{index} = a_{k-1}$; // $s_{index} = a_k$ in the discontinuous case
  i =k-1; // i =k in the discontinuous case
End

Fig. 1. SW segmentation algorithm.

noises. Although the SW method is an online method and has low computation complexity, it is criticized for lacking a global view of the whole time series, resulting in an approximation that is not as good as that of offline methods.

More recently, Keogh et al. have proposed the SWAB method in [15]. Its basic idea is described as follows: First, SWAB takes as its input a data sequence of user-specified length, which is called an *SW*. Then, it applies the bottom-up method to divide this sequence into a few segments. The leftmost segment is taken out as a new finished segment, and the data points in other segments together with the new incoming data points comprise the new window. The SWAB method segments the online data sequence by continuously applying the bottom-up method to the new window and taking out the leftmost segment as the finished segment. This method makes use of the low representation error of the bottom-up segmentation method and adapts it to an online method. This method gives a lower representation error at the cost of more segments.

While the vast majority of research has focused on representations that are calculated offline and represent each value with approximately equal fidelity, Palpanas et al. [21] have proposed a method to allow the online amnesic approximation of streaming time series, which represents the recent past with greater precision. Therefore, it is a better fit for practical needs. This method generates discontinuous representation due to linear regression approximation.

3 Novel Online Segmentation Algorithms: FSW and SFSW

In this section, we first present the new segmentation criterion that reduces the time complexity of the classic SW method by making use of the maximum vertical distance (MVD) measure, and then, we introduce the proposed two novel online segmentation algorithms, the FSW and SFSW methods.

3.1 Maximum Vertical Distance and Segmentation Criterion

All segmentation algorithms require certain measures to evaluate the goodness of fit for a potential segment. The residual error is such a measure, which is commonly used in conjunction with linear regression [15]. It is simply the sum of the squares of vertical distances between actual data points and the best fit line. Another measure is the MVD between the data points and the best fit line, which is commonly used in linear interpolation [6], [15]. FSW and SFSW adopt the MVD measure since they are both based on the linear interpolation model. MVD is used as the error bound to determine whether one more data point could be added to the current segment.

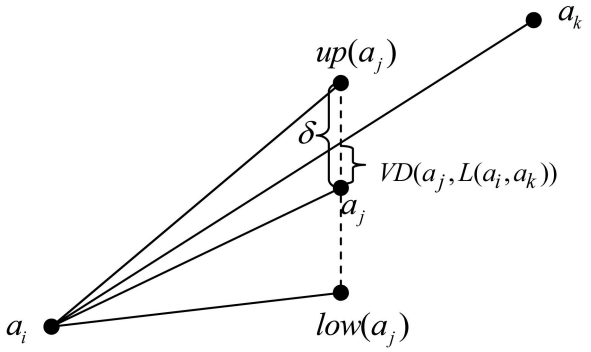

Fig. 2. Geometric interpretation of Property 1.

In the classic SW method, both residual error and MVD can be implemented to evaluate how well a potential segment fits. Its basic structure is given in Fig. 1, where *Calerror* is a function to calculate the error of the current segment. It works by initializing the first data point of a time series to be the left endpoint of a potential segment and then attempting to put one more data point into the segment at each step under a certain error calculation method. In the continuous approximation case, the endpoint of the previous segment is the beginning point of the next segment.

In the classic SW method, each time a new data point arrives, both measures have to recompute all the data points in the potential segment due to the change of the fit line. We found that this recomputation is unnecessary when we use the MVD measure. We propose a new segmentation criterion that improves the efficiency of the classic SW method by computing each data point only once.

For a time series $T = (\ldots, a_i, \ldots, a_j, \ldots)$ and $\delta > 0$, where δ is the user-specified maximum error tolerance and $a_j = (x_j, y_j)$ is a real point in R, we denote by $L(a_i, a_j)$ the line passing through a_i and a_j and $l(a_i, a_j)$ the slope of $L(a_i, a_j)$. Let $up(a_j) = (x_j, y_j + \delta)$ and $low(a_j) = (x_j, y_j - \delta)$. It is evident that $l(a_i, low(a_j)) < l(a_i, up(a_j))$. $VD(a, L(a_i, a_j))$ is defined as the vertical distance of data point a to line $L(a_i, a_j)$. With these notations, the slope of the approximation line satisfies the following property.

Property 1. *For a time series* $T = (\ldots, a_i, \ldots, a_j, \ldots, a_k, \ldots)$ *and* $\delta > 0$, $VD(a_j, L(a_i, a_k)) \leq \delta$ *iff* $l(a_i, low(a_j)) \leq l(a_i, a_k)) \leq l(a_i, up(a_j))$.

This property is simply illustrated in Fig. 2. The interpolating line is passing through a_i and a_k. For any point a_j between a_i and a_k, we need to determine whether the vertical distance of a_j to the interpolating line, $VD(a_j, L(a_i, a_k))$, is within δ. In Fig. 2, we can see that in order to make $VD(a_j, L(a_i, a_k)) \leq \delta$ hold, $L(a_i, a_k)$ must lie

between the lines $L(a_i, up(a_j))$ and $L(a_i, low(a_j))$, which is equivalent to $l(a_i, low(a_j)) \le l(a_i, a_k)) \le l(a_i, up(a_j))$, and vice versa. Based on this property, we propose our segmentation criterion as follows:

Segmentation criterion. For the current segment $S = (a_i, \ldots, a_j)$ and $\delta > 0$, let $lowl_{(i:j)} = \max_{i<t\le j} l(a_i, low(a_t))$ and $upl_{(i:j)} = \min_{i<t\le j} l(a_i, up(a_t))$. The newly arriving data point a_{j+1} can be added into the segment S iff $lowl_{(i:j)} \le l(a_i, a_{j+1}) \le upl_{(i:j)}$.

This segmentation criterion is derived directly from Property 1:

$$\begin{aligned} lowl_{(i:j)} \le l(a_i, a_{j+1}) \le upl_{(i:j)} \\ \Leftrightarrow \max_{i<t\le j} l(a_i, low(a_t)) \le l(a_i, a_{j+1}) \le \min_{i<t\le j} l(a_i, up(a_t)) \\ \Leftrightarrow l(a_i, low(a_t)) \le l(a_i, a_{j+1}) \le l(a_i, up(a_t)),\ i < t \le j \\ \Leftrightarrow VD\big(a_t, L(a_i, a_{j+1})\big) \le \delta,\ i < t \le j \quad \text{(Property 1)} \\ \Leftrightarrow a_{j+1} \text{ can be added to the segment } S. \end{aligned}$$

According to our segmentation criterion, each time a new data point arrives, we do not need to compute all the vertical distances between the data points and the new interpolating line to get the MVD. Instead, we simply compare the slope of the new line with the *lowl* and *upl* of the current segment and update them accordingly. It is an incremental processing technique. Thus, we have our first result: the computational complexity of the classic SW method can be reduced by using this segmentation criterion. A detailed complexity analysis will be given in Section 4.

3.2 FSW Method

Based on the segmentation criterion proposed in Section 3.1, a novel online segmentation method, FSW, is proposed in this paper. We call a point a Candidate Segmenting Point (CSP) if it may be chosen to be the next eligible segmenting point, i.e., the distances of all the points lying between the last segmenting point and the new chosen one to the new interpolating line are all within the maximum error tolerance. The key idea of FSW is to search for the farthest CSP to make the current segment as long as possible under the given maximum error tolerance.

According to the segmentation criterion, each time a new point arrives, the new updated interval $[lowl, upl]$ is contained in the old one. We call the interval $[lowl, upl]$ a *feasible space*. The feasible space is nonempty, which implies that it is possible for future points to be CSPs, even though the current new point is not a CSP. For example, in Fig. 3, the vertical distance of a_2 to $L(a_1, a_3)$ exceeds δ because $l(a_1, a_3)$ is outside of the interval $[l(a_1, low(a_2)), l(a_1, up(a_2))]$, so a_3 is not a CSP. However, a_4 is a CSP because $l(a_1, a_4)$ falls into the feasible space $[\max_{t=2,3}(l(a_1, low(a_t))), \min_{t=2,3}(l(a_1, up(a_t)))]$, which is the shaded area in Fig. 3, i.e., the vertical distances of points a_2 and a_3 to $L(a_1, a_4)$ are within the maximum error tolerance δ. The feasible space shrinks as the new points arrive. When the feasible space becomes empty, i.e., $upl < lowl$, it is impossible for any future data point to be a CSP. Therefore, if we choose the farthest CSP to be the next segmenting point, the current segment is certainly made the longest.

Finally, the FSW algorithm is given in Fig. 4.

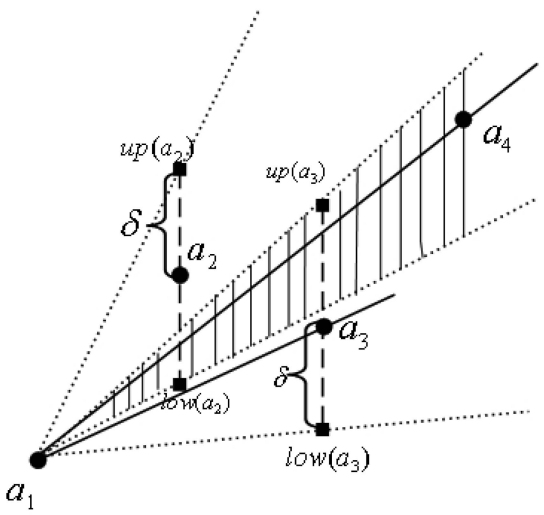

Fig. 3. Illustration of feasible space of $L(a_1, a_4)$.

3.3 SFSW Method

The SW method is often criticized for lacking an overall view of the whole time series, which results in the incoming data points having no influence on the finished segments [15], [24]. To overcome this deficiency, we propose another online segmentation method, the SFSW segmentation algorithm. The idea of this method involves backward segmenting one segment to refine the previously obtained endpoint every time we obtain two forward segments using the FSW method.

The introduction of the stepwise strategy to the FSW method stems from our intuition of the ideal time series. Suppose an ideal time series T itself is piecewise linear; $(s_1, s_2, s_3, \ldots)$ are the real segmenting points, which are marked by black points in Fig. 5. If we use the FSW method, due to the maximum error tolerance, we always find the point s'_i $(i = 2, 3, \ldots)$ as the segmenting points. Suppose we have achieved two segments S_1 and S_2 by the FSW segmentation method starting from point s_1. Then, we backward segment the sequence starting from the endpoint s'_3 to get the segment S_b and the backward segmenting point s_b. Intuitively, we believe that the real segmenting point should lie between the forward segmenting point s'_2 and the backward segmenting point s_b. After we find the real segmenting point s_2, we continue the stepwise process starting from that point.

However, a time series is seldom in such a neat linear form, and errors are inevitable, so we plot the stepwise process in Fig. 6 in a more general way. Suppose from the starting point a_i, two segments $S_1 = (a_i, \ldots, a_f)$ and $S_2 = (a_f, \ldots, a_j)$ are obtained by using the FSW method. Then, we segment the time series data from the last point a_j in reverse order and get segment $S_{b1} = (a_j, a_{j-1}, \ldots, a_b)$, and the remaining points $(a_b, a_{b-1}, \ldots, a_i)$ make up segment S_{b2} (see Fig. 6).

We claim that $b \le f$. From the forward segmenting process beginning with a_f, we obtained the segment S_2 with endpoint a_j. That means that a_j is the farthest point that can be added into the segment S_2. Therefore, the distance of any point between a_f and a_j to the interpolating

Algorithm 3.2 Feasible Space Window Segmentation Algorithm (FSW)

Input: time sequence $(a_1, a_2, \cdots a_n, \cdots)$, max_error threshold δ

Output: segmenting points $(s_1, s_2, \cdots, s_k, \cdots)$

Initial: $i = seg\_no = csp\_id = 1$, $s_1 = a_1$, $lowl = -\infty$, $upl = +\infty$

While not finished segmenting time series

$i = i + 1$;

$upl = \min(upl, l(s_{seg\_no}, up(a_i)))$, $lowl = \max(lowl, l(s_{seg\_no}, low(a_i)))$;

If $upl < lowl$ **Then**

$seg\_no = seg\_no + 1$;

$s_{seg\_no} = a_{csp\_id}$; // the farthest CSP is a new segmenting point.

$i = csp\_id$;

$lowl = -\infty$; $upl = +\infty$; //reset

Else

If $lowl \le l(s_{seg\_no}, a_i) \le upl$ **Then**

$csp\_id = i$; // record ID of farthest CSP.

End

End

End

Fig. 4. FSW segmentation algorithm.

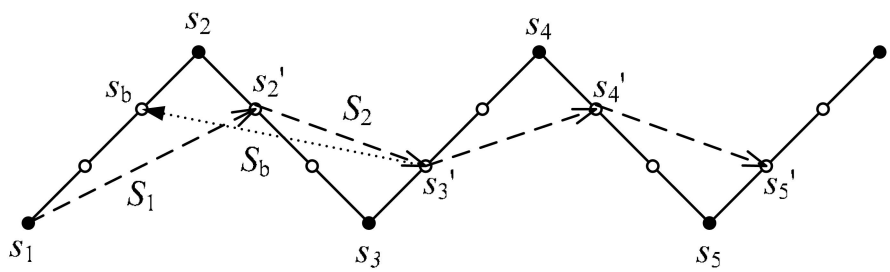

Fig. 5. Intuitions behind the stepwise strategy.

line $L(a_f, a_j)$ is less than the maximal vertical distance δ. In the backward segmentation process from a_j, the interpolating line $L(a_j, a_f)$, which is the same one to $L(a_f, a_j)$, and from the forward segmenting process, we know the distance of any point between a_f and a_j to $L(a_j, a_f)$ is less than δ. By definition, a_f is a feasible candidate endpoint for the reverse segment starting with a_j. Furthermore, as the FSW method always finds the farthest point that can be added into the segment for a given starting point, the backward segmenting point a_b is the farthest feasible segmenting point for segment S_{b1} starting with a_j. It follows that $b \le f$.

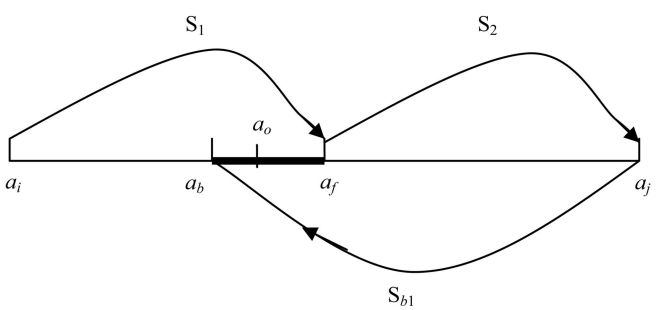

Fig. 6. Stepwise segmenting process.

In the forward segmenting process, the points $(a_b, \ldots, a_f)$ are segmented into the same segment S_1 with the points $(a_i, \ldots, a_{b-1})$, while in the backward segmenting process, they are segmented into the same segment S_{b1} with the points $(a_{f+1}, \ldots, a_j)$. That means that the only uncertainty lies in the adscription of the points $(a_b, \ldots, a_f)$. Heuristically, each data point a_t $(b \le t \le f)$ is possibly a segmenting point, which separates the data sequence $(a_i, \ldots, a_j)$ into two segments. In our algorithm, the point between $(a_b, \ldots, a_f)$ is chosen as the final optimal segmenting point for the sequence $(a_i, \ldots, a_j)$, which drops the sum of representing errors of both segments to the smallest while keeping the MVD for each segment under the user-specified tolerance. Hopefully, we choose a "more reasonable" segmenting point than the segmenting point a_f we obtained in the forward segmenting process. Suppose the optimal segmenting point is a_o $(b \le o \le f)$, the new finished

Algorithm 3.3 Stepwise FSW Segmentation Algorithm (SFSW)

Input: time sequence $(a_1, a_2, \cdots, a_n, \cdots)$, max_error threshold δ

Output: segmenting points $(s_1, s_2, \cdots, s_k, \cdots)$

Initial: i=1, *index*=1, $s_1 = a_1$

While not finished segmenting time series

index = *index* + 1;

$[a_f, a_j] = Swf(i)$;

$a_b = Swb(j)$;

$o = \text{Arg}\min_{b \le k \le f} \{Sre[\text{S}(a_i, \cdots, a_k) + \text{S}(a_k, \cdots, a_j)]\}$;

$s_{index} = a_o$;

$i = o$;

End

Fig. 7. SFSW segmentation algorithm.

segment is thus $(a_i, \ldots, a_o)$. Clearly, it is the data points coming after a_f that cause the change in the segmenting point of the previously finished segment S_1. Next, we start with a_o and repeat the above process until the entire time series has been handled. Therefore, unlike the SW method in which only the past points are used to determine the segmenting place and the points after the segmenting point have no influence on the previously obtained segment, our stepwise method uses the backward segmenting process to revise the previous segmenting point. That means that the segmenting point is determined by the points both before and after it. Although unlike the bottom-up method, which determines the segmenting place from the whole sequence, our method still refines the segmenting point locally. The structure of the SFSW algorithm is given in Fig. 7. In the algorithm, the *Swf* function performs two forward segmentations using the FSW method starting from data point a_i and returns two segmenting points a_f and a_j. *Swb* is the function that backward segments the data points from a_j and returns the backward segmenting point a_b. *Sre* is the function that calculates the sum of representation errors of the segments. a_o $(b \le o \le f)$ is the optimal segmenting point that makes *Sre* the smallest. Then, we begin the stepwise segmenting process from data point a_o.

4 Complexity Analysis

To simplify the complexity analysis, we apply the online methods to a time series with fixed length. For a given time series T of n data points, we denote the number of segments and the average segment length by K and L, respectively; thus, $n = K * L$. However, there is a problem with this assumption for online segmentation methods. Although the first data point of T is always a segmenting point, the last one may not be. For the sake of simplicity, we avoid this problem by simply requiring the last data point of time series T to be a segmenting point. Since this restriction increases K by at most one and K is generally much larger than one, the impact of the restriction on the results of segmentation algorithms is small enough to be omitted.

In particular, the analyses of FSW and SFSW algorithms benefited from the above restriction. For FSW, in the worst case, theoretically, it is possible for the feasible space to stay nonempty even if a new CSP never arrives. For example, in Fig. 8, suppose the current segmenting point is a_1; if every data point $a_i (i = 3, \ldots)$ is not a CSP but is very close to the straight line $L(a_1, low(a_2))$, i.e., $VD(a_i, L(a_1, a_2)) > \delta$, the feasible space will surely stay nonempty. When the feasible space finally becomes empty after searching through large numbers of non-CSPs, the next segmenting point is just a_2. Although it hardly happens in practice, we can still make sure that the computing time of the FSW algorithm is under control by applying this restriction to it. The same analysis applies to the SFSW algorithm.

We list below the complexity analyses of FSW and SFSW algorithms, as well as the complexities of classic SW and SWAB. The standard notation $O(f(n))$ and $\theta(f(n))$ are adopted here to represent respectively the upper bound and both lower and upper bounds of computational complexity.

SWAB. The SWAB method can be regarded as a combination of the Bottom-Up method and the SW method. The SWAB method keeps an SW of length w, which is initially set to be the length of five or six segments in [15]. At the beginning, SWAB applies the Bottom-Up algorithm to the SW and sets the leftmost segment as the first segment it produced. Then, the data points in the first segment are deleted from the window, and new data points are added in. The same process is repeated to produce the second segment and so on. We can easily see that the time complexity of SWAB is a small constant factor higher than that of the Bottom-up algorithm by setting the length of the SW as a constant times the average segment length L. In [15], the authors have reported that the time complexity of the Bottom-Up algorithm is $O(Ln)$. Therefore, the time complexity of SWAB is $O(Ln)$.

Classic SW. For this algorithm, each time a new point arrives, we have to determine whether or not to put this point into the current segment by remeasuring the

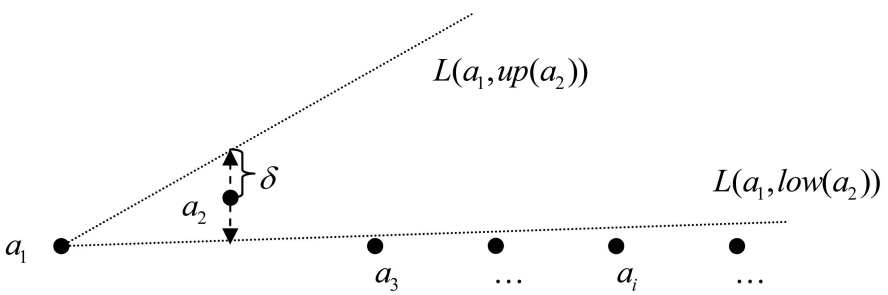

Fig. 8. The worst case of FSW.

TABLE 1
Summary of Time Complexities of Online Segmentation Methods

Algorithm	Complexity
ISW	$O(n)$
Classic SW	$O(Ln)$
SWAB	$O(Ln)$
FSW	$O(Kn)$
SFSW	$O(Kn^2)$

maximum distance. Since the length of a potential segment grows from 2 to L and we do not need to measure the endpoints of the segment, the time to compute a single segment is $\sum_{i=1}^{L-1}\theta(i) = \theta(L^2)$. Therefore, the time complexity is $K \cdot \theta(L^2) = \theta(Ln)$ or $O(Ln)$.

Based on the proposed segmentation criterion, the classic SW method can in fact determine whether or not a new point a_i should be put into the current segment by simply comparing the slope of the new line $L(a_i, a_t)$ (supposing a_i is the left endpoint of the current segment) with the feasible space. That is, we put a_t into the current segment if and only if $l(a_i, a_t)$ is in the feasible space. After that, we should also update the feasible space as FSW does. Evidently, the time for handling a new data point is constant. Therefore, by using this incremental technique, the time complexity of the classic SW method is reduced to $O(n)$. We call this new version of classic SW the Incremental SW (ISW) method. Since the only difference between ISW and classic SW is in the time complexity, we will not distinguish them in the remaining part of this paper unless otherwise specified.

FSW algorithm. In the worst case, each time FSW searches for the next segmenting point, it has to reach the last data point of time series T, which implies that the complexity of finding one segmenting point is $O(n)$. It has been supposed that there are K segments in time series T, and therefore, the complexity of the FSW algorithm is $O(Kn)$.

SFSW. Suppose SFSW has produced i $(i=0,1,\ldots,K-1)$ segments so far. To produce the $(i+1)$th segment, SFSW has to perform two forward searches and one backward search by applying the FSW algorithm to the remaining N_i $(N_i = n - iL)$ data points and then choose the optimal one to be the next segmenting point. In the worst case, the time complexity of the two forward searches is $O(2N_i)$, and the time of the backward search is $O(N_i)$. The time of the choosing procedure is $O(N_i^2)$, since the maximum number of data points to be measured in this procedure is N_i. In total, the time complexity of producing the $(i+1)$th segment is $O(2N_i + N_i + N_i^2) = O(N_i^2)$. Therefore, the time complexity of SFSW is $\sum_{i=0}^{K-1} O(N_i^2) = O(\sum_{i=0}^{K-1}(n - iL)^2) = O(Kn^2)$.

We summarize this complexity study in Table 1. The results are achieved based on a complete sequence of length n. Since the L and K yielded by the different segmentation algorithms may be very different for the same data sequence of length n, we tested the runtime of the algorithms experimentally in the next section to show the computing efficiency of the algorithms.

5 Experiments

In this section, we provide extensive empirical comparisons of our methods, FSW and SFSW, with the classic SW method and the SWAB method. Keogh et al. [15] have done experiments on many types of time series. Their results show that in most test cases, the bottom-up method achieved a more precise representation under the same maximum error tolerance. The SWAB method is an online method that has gotten comparable results to the Bottom-up method. Before the comparison, we should specify again that the goal of our segmentation is to obtain as few segments as possible under the user-specified maximum error tolerance. Therefore, in order to evaluate the segmentation of a given time series, we consider the number of segments, as well as the representation error.

5.1 Experiment Settings

The experiments are conducted on 10 time series data sets from different fields, including finance, medicine, astronomy, and industry. They include Tickwise, ERP, Phone1, ECG_Heart_Rate, Network, Shuttle, Wind, Powerplant, Memory, and Speech and are taken from the UCR Time Series Data Mining Archive [11].

It has been shown in [15] that the performance of the algorithms is influenced by the setting of *max_error*. For example, when *max_error* is very small, all algorithms give $n-1$ segments (any two adjacent data points form a segment in the interpolation approximation case). When *max_error* becomes positive infinite, all algorithms give only one segment. In our experiments, we use the relative *max_error*, the percentage of *max_error* to the range of time series values, called the Maximum Error Percentage (MEP), instead of the absolute *max_error*. This is because the same absolute *max_error* has different influences on time series with different ranges. For example, the maximum error 2 is acceptable for the time series whose values vary from 1 to 100 but unacceptable for the time series whose values vary from 10 to 12. If MEP is set as 20 percent for the time series whose values vary from 10 to 20, then in the algorithms, the *max_error* $\delta = (20-10) * 20\ \text{percent} = 2$.

Furthermore, since the sensitivities of the time series to MEP are not the same, we have to choose appropriate MEP values for each data set. The objective of a careful choice is to make sure that the numbers of segments are in a reasonable range so as to avoid overfragmentation and very coarse segmentation cases. In our experiment, the average segment length L is at least two. The representation error for a segmentation is measured by the total sum of residual errors of all segments, denoted by RE. In the SWAB algorithm, to obtain the best window length setting, we first achieve the classic SW results and then set the window length of SWAB as five times the average segment length of classic SW, which is the optimal setting mentioned in [15]. In the following experiments, we set five different MEP values for each data set and apply the methods to the 10 data sets with these five MEPs. The segmentation results are analyzed in the following sections. In the experiments,

TABLE 2
Average Normalized Number of Segments of Different Methods

SM	SW	SWAB	FSW	SFSW
Tickwise	1	1.17	0.69	0.65
ERP	1	1.34	0.68	0.61
Phone1	1	1.03	0.70	0.67
ECG_Heart_Rate	1	1.00	0.30	0.26
Network	1	1.08	0.65	0.64
Shuttle	1	1.06	0.20	0.18
Wind	1	1.00	0.96	0.96
Powerplant	1	1.13	0.76	0.76
Memory	1	1.34	0.81	0.73
Speech	1	1.36	0.88	0.85
Average	1	1.15	0.66	0.63

TABLE 3
Average Normalized Representation Error of Different Methods

SM	SW	SWAB	FSW	SFSW
Tickwise	1	0.51	0.95	0.74
ERP	1	0.53	0.90	0.66
Phone1	1	0.63	1.25	1.09
ECG_Heart_Rate	1	0.90	1.77	1.83
Network	1	0.52	1.37	1.03
Shuttle	1	0.63	1.48	1.43
Wind	1	0.84	1.04	0.85
Powerplant	1	0.41	0.88	0.75
Memory	1	0.31	0.93	0.66
Speech	1	0.38	0.98	0.70
Average	1	0.57	1.15	0.97

the result of the SW method is used as the benchmark. The symbols used in the experiments are defined as follows:

- SM: segmentation methods,
- SW: classic sliding window method,
- SWAB: sliding window and bottom-up method,
- FSW: feasible space window method,
- SFSW: stepwise feasible space window method,
- MEP: maximal error percentage of the value range,
- RE: representation error (total sum of residual errors of all segments),
- SN: number of segments,
- NRE: normalized representation error; $\mathrm{NRE(SM)} = \mathrm{RE(SM)/RE(SW)}$, and
- NSN: normalized number of segments; $\mathrm{NSN(SM)} = \mathrm{SN(SM)/SN(SW)}$.

5.2 Results of the Number of Segments

In this experiment, we compare the numbers of segments produced by the different methods. The criterion is that the fewer the segments, the better the segmentation. After the segmentation result is obtained by applying a segmentation method to a time series, we compute the average of the normalized numbers of segments corresponding to the five MEPs and fill the number in the corresponding cell of Table 2. By definition, the smaller the NSN is, the better the segmentation is. The bottom row shows the averages for the 10 time series, which represents the "average" performance of a method on the number of segments. In Table 2, we can see that both of our proposed methods dramatically reduce the number of segments and that SWAB results in the largest number of segments. In addition, the results of SFSW are a little better than those of FSW, since the goal of the stepwise strategy is only to reduce the representation error. All the results confirm the initial ideas of our algorithms.

5.3 Results of Representation Errors

In this part, we investigate the representation errors of our algorithms. We applied each method to each of the time series and computed the average of the normalized representation errors corresponding to the five MEPs and filled the numbers in the cells of Table 3. By definition, the smaller the NRE is, the better the segmentation. The bottom row shows the averages for the 10 time series. In Table 3, it can be seen that SWAB always reduces the representation errors at the cost of additional segments. What is most surprising is that in most cases, FSW and SFSW outperformed SW by reducing the representation error with fewer segments.

5.4 Overall Performances of Segmentation Methods

The goal of the segmentation is to minimize the number of segments and the representation error at the same time. However, in general, when the segments are few, the representation error will be large. Conversely, when the representation error is small, the number of segments will probably be large. To evaluate these methods in terms of both the representation error and the number of segments, the product of the normalized number of segments and the normalized representation error is adopted as the measurement, denoted by OE. When both the number of segments and the representation error are very small or large, the product will be low or high, respectively. When one factor is small and the other is large, the product will be a moderate value. Heuristically, this can be used to evaluate the overall performance of the segmentation.

The results are listed in Table 4. The value contained in each cell is the product of the values contained in the cells of the same position in Tables 2 and 3, representing the "average" overall performance of a method on a time series. By definition, the smaller the OE is, the better the segmentation is. The bottom row shows the average for the 10 time series. Table 4 shows that the overall performance of SFSW is better than that of FSW and SW in all test cases and better than SWAB in most cases except for four (Network, Powerplant, Memory, and Speech).

TABLE 4
Overall Performance Evaluation

SM	SW	SWAB	FSW	SFSW
Tickwise	1	0.60	0.66	0.48
ERP	1	0.72	0.61	0.40
Phone1	1	0.65	0.78	0.59
ECG_Heart_Rate	1	0.90	0.55	0.49
Network	1	0.56	0.89	0.65
Shuttle	1	0.67	0.31	0.28
Wind	1	0.83	0.99	0.81
Powerplant	1	0.39	0.66	0.50
Memory	1	0.41	0.75	0.47
Speech	1	0.50	0.87	0.60
Average	1	0.62	0.71	0.53

5.5 Experiments on Random Walk

We also experimented on an artificial time series, Random Walk, which includes about 50,000 data points and is from the UCR Time Series Data Mining Archive [11]. This kind of time series is often used to simulate stock prices. The goal of this experiment is to use one time series as an example to show more detailed results with the parameter MEP.

The MEPs are set by $0.5 \text{ percent} * i \ (i = 1, \ldots, 20)$, which range from 0.5 percent to 10 percent. When the MEP is 0.5 percent, the random walk series is divided into about 6,000 segments, and each segment contains an average of seven data points. When the MEP is 10 percent, the random walk series is divided into about 25 segments, and each segment contains an average of 2,000 data points. The chosen numbers of segments are reasonable.

Fig. 9 shows the segmentation results. Fig. 9a shows the normalized numbers of segments of each method for different MEPs, Fig. 9b shows the normalized representation errors of each method for different MEPs, and Fig. 9c shows the overall performances of each method for different MEPs. In Fig. 9a, we can see that the numbers of segments achieved by the FSW and SFSW methods are far less than the numbers of segments achieved by SW and SWAB. In Fig. 9b, we can see that except for $\text{MEP} = 0.5$ percent for the FSW method, all the other methods produced a lower representation error than the SW method. Fig. 9c shows that for the random walk series, SFSW's overall performance is far better than that of SWAB and FSW. These results are consistent with the results of the previous experiments.

5.6 Experiments on Runtime

In Section 4, we have shown that almost all of the algorithms analyzed are linear with n except that SFSW is $O(Kn^2)$. However, we cannot conclude that SFSW is the slowest or that all the others are about the same level of efficiency. In fact, the computing the efficiency of the algorithms may vary significantly due to the different constant factors. More importantly, the L and K yielded by the different segmentation algorithms may be very different for the same data sequence of length n, which may also result in different running performance. Therefore, evaluating the efficiency of the algorithms empirically is undoubtedly meaningful, especially for the online algorithms. We vary the MEP from 1 percent to 20 percent with an interval of 1 percent and record the runtime of each algorithm. For all of the MEPs, the average runtime of the algorithms for each time series are summarized in Table 5 to illustrate their rough running performance. The

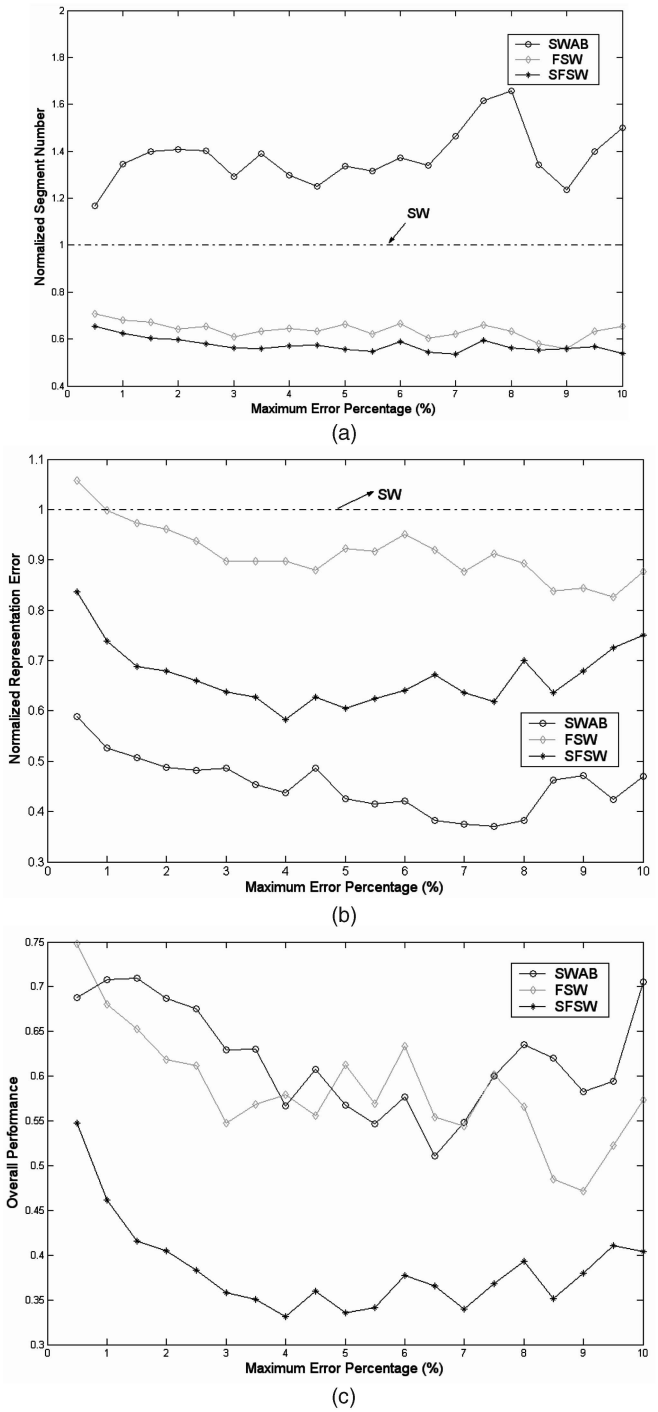

Fig. 9. (a) Normalized number of segments. (b) Normalized representation errors. (c) Overall performance evaluations.

TABLE 5
Average Runtime of Algorithms (in Seconds)

ME	SW	SWAB	FSW	SFSW
Tickwise	25.9466	2474.4037	0.0132	75.6792
ERP	0.6546	89.5632	0.0172	1.5806
Phone1	0.0041	0.0583	0.0033	0.0125
ECG_Heart_Rate	0.0025	0.0461	0.0033	0.0248
Network	6.3643	117.6865	0.0025	4.6233
Shuttle	0.0016	0.0231	0.0017	0.0057
Wind	0.6119	6.1052	0.0017	0.0493
Powerplant	0.0082	0.6004	0.0000	0.0115
Memory	0.0395	5.8413	0.0025	0.0921
Speech	0.0016	0.0576	0.0000	0.0000
Average(std)	3.3635 (8.1745)	269.4385 (775.9328)	0.0045 (0.0058)	8.2079 (23.7523)

hardware environment is CPU Celeron 2.4, 512-Mbyte memory, and 80-Gbyte hard disk. The programs are written in C++ and run in Windows XP.

In Table 5, the FSW method is the fastest one, as we expected, followed by SW and SFSW, and SWAB is the slowest. SFSW is slower than FSW due to the stepwise strategy. Also, the standard deviation (Std) values in the last row indicate that the FSW method is more reliable in terms of computing time cost than the others.

6 Conclusions

In this paper, we first proposed a new segmentation criterion that successfully reduces the computational complexity of the classic SW method from $O(Ln)$ to $O(n)$. Based on this criterion, we developed two novel online segmentation methods: the FSW method attempts to reduce the number of segments by searching for the farthest endpoint of a potential segment along the time direction; the SFSW method is a combination of the FSW method and the *stepwise* strategy, which refines the segmenting points by taking into account the effects of new incoming points so that the representation error can be reduced.

The extensive numeric experiments we performed demonstrate the advantages of each algorithm. The FSW and SFSW methods always generate far fewer segments than the other two methods, given the same MEP. In addition, the SWAB method produces the results with the lowest representation error. If we take into account the number of segments and representation error together using the OE metric, SFSW outperforms the other methods. However, from the aspect of computing efficiency, the FSW method is the fastest and the most reliable.

There are a number of possible research directions for future study. Foremost among them, an amnesic representation is an interesting topic since it comes closer to reflecting actual practice. This would mean that recent data points would be approximated more precisely. The representation fidelity is reduced with the passing of time. We will try to refine our method in this direction in further studies. Second, in the SFSW method, only one backward step is performed. This method could be generalized to a (p, q) stepwise method in the future, where p is the forward segmentation steps, q is the backward steps, and $q < p$. Third, we have done some work in continuous feature discretization of static data sets [20] to divide the value range of the continuous feature into several intervals. We could apply the ideas from both our segmentation methods to the discretization method. So far, we have only discussed segmentation methods for a one-dimensional time series sequence. However, some time series databases collect multidimensional time series. For example, in weather monitoring, each point is associated with temperature, humidity, wind speed, and so forth. When segmenting such a time series, we need to consider not only the temporal relation of the points but also the relation between different dimensions of the points. The last but most important point is that since segmentation is just a preprocessing step for time series mining, how to integrate it into real-world time series mining tasks will be our next research target.

Acknowledgments

This work was supported by UGC Research Grant CityU 120805 from the Hong Kong Government and the CityU SRG Grant 7001805.

References

[1] R. Agrawal, C. Faloutsos, and A. Swami, "Efficient Similarity Search in Sequence Databases," *Proc. Fourth Conf. Foundations of Data Organization and Algorithms (FODO '93),* pp. 69-84, 1993.

[2] U. Appel and A.V. Brandt, "Adaptive Sequential Segmentation of Piecewise Stationary Time Series," *Information Science,* vol. 29, no. 1, pp. 27-56, 1983.

[3] G.F. Bryant and S.R. Duncan, "A Solution to the Segmentation Problem Based on Dynamic Programming," *Proc. Third IEEE Conf. Control Applications (CCA '94),* pp. 1391-1396, 1994.

[4] F.K.-P. Chan, A.W.-C. Fu, and C. Yu, "Haar Wavelets for Efficient Similarity Search of Time-Series: With and without Time Warping," *IEEE Trans. Knowledge and Data Eng.,* vol. 15, no. 3, pp. 686-705, May/June 2003.

[5] S.R. Duncan and G.F. Bryant, "A New Algorithm for Segmenting Data from Time Series," *Proc. 35th IEEE Conf. Decision and Control (CDC '96),* pp. 3123-3128, 1996.

[6] T.C. Fu, F.L. Chung, V. Ng, and R. Luk, "Evolutionary Segmentation of Financial Time Series into Subsequences," *Proc. Congress on Evolutionary Computation (CEC '01),* pp. 426-430, 2001.

[7] X. Ge and P. Smyth, "Segmental Semi-Markov Models for Endpoint Detection in Plasma Etching," *IEEE Trans. Semiconductor Eng.,* 2001.

[8] A. Gionis and H. Mannila, "Segmentation Algorithms for Time Series and Sequence Data," *Tutorial in SIAM Int'l Conf. Data Mining,* 2005.

[9] S. Guha, N. Koudas, and K. Shim, "Data-Streams and Histograms," *Proc. 33rd Ann. ACM Symp. Theory of Computing (STOC '01),* pp. 471-475, 2001.

[10] http://www.nyse.com/taq/, 2006.

[11] E. Keogh and T. Folias, *The UCR Time Series Data Mining Archive,* Computer Science and Eng. Dept., Univ. of California, http://www.cs.ucr.edu/~eamonn/TSDMA/index.html, 2002.

[12] K.V.R. Kanth, D. Agrawal, and A.K. Singh, "Dimensionality Reduction for Similarity Searching in Dynamic Databases," *Proc. ACM SIGMOD '98,* pp. 166-176, 1998.

[13] E. Keogh and P. Smyth, "A Probabilistic Approach to Fast Pattern Matching in Time Series Databases," *Proc. ACM SIGKDD '97,* pp. 20-24, 1997.

[14] E. Keogh et al., "Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases," *J. Knowledge and Information Systems,* vol. 3, no. 2, pp. 263-286, 2001.

[15] E. Keogh et al., "Segmenting Time Series: A Survey and Novel Approach," *Data Mining in Time Series Databases,* second ed. World Scientific, 2003.
[16] E. Keogh and M. Pazzani, "An Enhanced Representation of Time Series Which Allows Fast and Accurate Classification, Clustering and Relevance Feedback," *Proc. ACM SIGKDD '98,* pp. 239-241, 1998.
[17] E. Keogh and M. Pazzani, "Scaling Up Dynamic Time Warping to Massive Dataset," *Proc. Third European Conf. Principles of Data Mining and Knowledge Discovery (PKDD '99),* pp. 1-11, 1999.
[18] A. Koski, M. Juhola, and M. Meriste, "Syntactic Recognition of ECG Signals by Attributed Finite Automata," *Pattern Recognition,* vol. 28, no. 12, pp. 1927-1940, 1995.
[19] L.C.-H. Lee, A. Liu, and W.-S. Chen, "Pattern Discovery of Fuzzy Time Series for Financial Prediction," *IEEE Trans. Knowledge and Data Eng.,* vol. 18, no. 5, pp. 613-625, May 2006.
[20] X.Y. Liu and H.Q. Wang, "A Discretization Algorithm Based on a Heterogeneity Criterion," *IEEE Trans. Knowledge and Data Eng.,* vol. 17, no. 9, pp. 1166-1173, Sept. 2005.
[21] T. Palpanas et al., "Online Amnesic Approximation of Streaming Time Series," *Proc. 20th Int'l Conf. Data Eng. (ICDE '04),* pp. 338-349, 2004.
[22] S. Park, S.W. Kim, and W.W. Chu, "Segment-Based Approach for Subsequence Searches in Sequence Databases," *Proc. 16th ACM Symp. Applied Computing (SAC '01),* pp. 248-252, 2001.
[23] D. Rafiei and A.O. Mendelzon, "Efficient Retrieval of Similar Time Sequences Using DFT," *Proc. Fifth Int'l Conf. Foundations of Data Organization (FODO '98),* pp. 249-257, 1998.
[24] H. Shatkay and S.B. Zdonik, "Approximate Queries and Representations for Large Data Sequences," Technical Report CS-95-03, Dept. of Computer Science, Brown Univ., 1995.
[25] H.J.L.M. Vullings, M.H.G. Verhaegen, and H.B. Verbruggen, "ECG Segmentation Using Time-Warping," *Advances in Intelligent Data Analysis,* pp. 275-285, 1997.
[26] B.K. Yi, H.V. Jagadish, and C. Faloutsos, "Efficient Retrieval of Similar Time Sequences under Time Warping," *Proc. 14th Int'l Conf. Data Eng. (ICDE '98),* pp. 201-208, 1998.
[27] Y. Zhao and S. Zhang, "Generalized Dimension-Reduction Framework for Recent-Biased Time Series Analysis," *IEEE Trans. Knowledge and Data Eng.,* vol. 18, no. 2, pp. 231-244, Feb. 2006.

Xiaoyan Liu received the MS degree in computational mathematics from Tsinghua University, China, in 2003 and the PhD degree in information systems from the City University of Hong Kong, Kowloon, in 2007. She is currently a senior research associate in the Department of Information Systems, City University of Hong Kong. Her research interests include data mining, financial engineering, scientific computing, business intelligent systems, and their applications. She is a member of the IEEE Computer Society.

Zhenjiang Lin received the BS degree in computational mathematics from Nankai University, China, in 1998 and the MS degree in computational mathematics from Tsinghua University, China, in 2003. He is currently a PhD candidate in the Department of Computer Science and Engineering, Chinese University of Hong Kong, New Teritorior, Hong Kong. His research interests include Web mining, scientific computing, and algorithm optimization.

Huaiqing Wang received the PhD degree in computer science from the University of Manchester in 1987. He is a professor in the Department of Information Systems, City University of Hong Kong, Kowloon. He specializes in research and development of business intelligence systems, intelligent agents, data mining, and their applications (such as intelligent financial systems, knowledge management systems, intelligent e-business systems, and conceptual modeling).