Multiscale Representations for Fast Pattern Matching in Stream Time Series

Xiang Lian, *Student Member*, *IEEE*, Lei Chen, *Member*, *IEEE*, Jeffrey Xu Yu, *Senior Member*, *IEEE*, Jinsong Han, *Member*, *IEEE*, and Jian Ma

Abstract—Similarity-based time-series retrieval has been a subject of long-term study due to its wide usage in many applications, such as financial data analysis, weather data forecasting, and multimedia data retrieval. Its original task was to find those time series similar to a pattern (query) time-series data, where both the pattern and data time series are static. Recently, with an increasing demand on stream data management, similarity-based stream time-series retrieval has raised new research issues due to its unique requirements during the stream processing, such as one-pass search and fast response. In this paper, we address the problem of matching both static and dynamic patterns over stream time-series data. We will develop a novel multiscale representation, called multiscale segment mean, for stream time-series data, which can be incrementally computed and thus perfectly adapted to the stream characteristics. Most importantly, we propose a novel multistep filtering mechanism, step by step, over the multiscale representation. Analysis indicates that the mechanism can greatly prune the search space and thus offer fast response. Furthermore, batch processing optimization and the dynamic case where patterns are also from stream time series are discussed. Extensive experiments show the multiscale representation together with the multistep filtering scheme can efficiently filter out false candidates and detect patterns, compared to the multiscale wavelet.

Index Terms—Similarity pattern match, stream time series, multiscale segment mean approximation.

1 Introduction

Recently, stream time-series data management has become a hot research topic due to its wide range of applications, such as Internet traffic analysis [13], sensor network monitoring [36], [21], moving object search [12], [24], and financial data analysis [31], which require continuously monitoring stream time series. Compared to traditional archived data, stream time series have their own characteristics: 1) data are frequently updated in stream time series. Thus, previous methods applied to archived data may not work in this scenario and 2) due to the frequent updates, it is very difficult to store all the data in memory or on disk, thus, data summarization and one-pass algorithms are usually required to achieve a fast time response.

In this paper, we deal with an important scenario in stream applications where incoming data are from a set of continuous stream time series and patterns from a set of static/dynamic time series. At each timestamp, a new data item is appended to each stream time series. We want to quickly find all the similar pairs up to the current time, one from stream time series and the other from patterns, so that their distances do not exceed a user-specified threshold ε. Later on, we will extend our approach to the dynamic case where patterns are also from a set of stream time series. Throughout this paper, we will use terms query pattern, pattern, and query time series, interchangeable with the same meaning.

One scenario of this problem is stock data monitoring. Given a set of real-time stock data and a number of predefined movement trends (e.g., "two bottom" or "head-shoulder" patterns), the system monitors the stock data and gives a fast response whenever the stock data are similar to the predefined patterns. In a weather warning system, a weather forecast station needs to determine the probability of storm at a certain location [18] based on the streaming sensor data. In particular, two sets of sensors are deployed to collect temperature and humidity data, respectively. In order to predict a storm in advance, the system needs to find temperature-humidity pairs from sensor data in the form of time series, whose movement trends (shapes) follow the storm pattern.

Another application can be found in coal mine surveillance [32], where sensors are deployed throughout channels in a coal mine to collect time-series data, such as the density of gas and temperature. For safety reasons, the monitoring system must detect dangers like gas leakage or fire alarm as early as possible. Since such dangers usually correspond to certain patterns in a contour map, we aim to find those stream time series (i.e., contour maps [32]) that match with some predefined (static) patterns, and warn the miners if any match is found. There are three tasks involving pattern matching over stream time series. First, we want to efficiently perform the pattern matching on the stream time series in order to detect possible danger as early as possible. Second, it is highly desired to perform fast pattern matching over a batch of series data (e.g., due to network congestions)

- *X. Lian, L. Chen, and J. Han are with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong. E-mail: {xlian, leichen, jasonhan}@cse.ust.hk.*
- *J.X. Yu is with the Department of System Engineering, The Chinese University of Hong Kong, William M.W. Mong Engineering Building, Shatin, NT, Hong Kong. E-mail: yu@se.cuhk.edu.hk.*
- *J. Ma is with the Nokia Research Center, Beijing 100176, P.R. China. E-mail: lian.J.Ma@nokia.com.*

Manuscript received 30 Mar. 2007; revised 4 Dec. 2007; accepted 12 Aug. 2008; published online 2 Sept. 2008.
Recommended for acceptance by S. Chakravarthy.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2007-03-0134.
Digital Object Identifier no. 10.1109/TKDE.2008.184.

at the same time. Third, since sensors are usually unreliable and each mine evacuation may cost millions of dollars, false alarms should be avoided. Thus, we can conduct the matching among multiple stream time series collected from spatially close sensors, where (dynamic) patterns are also from stream time series. Observing that spatially close sensors often report similar series data (i.e., having small distances between series), we can thus identify those malfunctioned sensors by their reporting data which are different from others and obtain more reliable data.

In fact, these scenarios of detecting similar patterns (e.g., "head-shoulder", storm, or fire patterns in applications mentioned above) over stream time series can be considered as a typical similarity search problem over time series. That is, given a time-series data set D, a pattern set Q, and a user-specified threshold ε, a similarity-search operation retrieves all the time series $t_i \in D$ such that for some pattern $q_j \in Q$, the distance between t_i and q_j does not exceed ε. This similarity search problem can be also extended to the cases where a batch of stream time-series data arrive at the same time and patterns are dynamic, which will be discussed later in this paper.

Note that previous works on similarity search on archived time series are not applicable to that over stream time series, since they cannot efficiently process frequent updates. Moreover, the existing work for searching on stream time-series data consider detecting a *single static* pattern over *multiple stream* time-series data [7], or checking which pattern (from *multiple static* patterns) is close to a *single stream* time series [16]. In contrast, our work focuses on the detection of *multiple static/dynamic* patterns over *multiple stream* time series.

In this paper, we propose a novel approach to efficiently perform the pattern matching over stream time series. Specifically, in order to save the computational cost and offer a fast response, we present a novel *multiscale* representation for time series, namely *multiscale segment mean* (MSM). We show that this representation can be incrementally computed, which is suitable for frequent update during stream processing. Furthermore, we show that our MSM representation works very well under all L_p-norms, for $p \geq 1$. This nice property offers the flexibility for users to choose different p, since different applications may need specific measures, for example, L_1-norm is robust against impulse noise [33] and L_∞-norm is suitable for atomic matching [2]. Most importantly, we propose a *multistep filtering* approach, with respect to the MSM representation, to prune false candidates before computing the real distances between patterns and stream time series. Note that, although the *segment mean* representation of a time series was studied in [33], MSM focuses on utilizing its *multiscale* property together with the multistep filtering to reduce the *computational cost* and maximize the *pruning power*.

To summarize, we make the following contributions in this paper:

1. We introduce a *multiscale* representation, namely, MSM, for stream time series, which is suitable for stream processing.
2. We propose a novel *multistep filtering* technique, called step-by-step (SS) scheme, on the MSM representation, and show that no *false dismissals* are introduced after filtering. Moreover, we give analytic results to show when we should use SS scheme and up to which scale we use it.
3. We present an efficient query procedure for pattern matching over stream time series. Moreover, we also discuss the batch processing optimization, and similarity matching with dynamic patterns.
4. We prove that, under L_2-norm, MSM has the same *pruning power* as another *multiscale* representation, *wavelets* (*Discrete Wavelet Transform* (DWT)), during the pattern matching; however, it is more powerful under other L_p-norms for $p \neq 2$, which has been confirmed by extensive experiments.

The rest of this paper is arranged as follows: Section 2 briefly reviews the related work. Section 3 formally defines our problem. Sections 4, 5, and 6 propose the MSM representation, multistep filtering technique, and query processing, respectively. Section 7 discusses the pattern matching with dynamic patterns. Section 8 evaluates the efficiency and effectiveness of our proposed methods. Finally, Section 9 concludes this paper.

2 Related Work

In this section, Section 2.1 overviews previous work on similarity search over archived time-series data. Then, Section 2.2 presents related works on monitoring over stream time series.

2.1 Similarity Search in Archived Time-Series Databases

In the research literature, many approaches have been proposed for the similarity search on the archived time series. The pioneering work by Agrawal et al. [1] proposed the *whole matching*, which finds data sequences of the same length that are similar to a query sequence. Later, Faloutsos et al. [15] extended this work to allow the *subsequence matching*, which finds subsequences in the archived time series that are similar to a given query series. In these two works, *Euclidean distance* is used to measure the similarity between (sub)sequences. In order to perform an efficient similarity search, the GEMINI framework [15] is proposed to index time series and answer similarity queries without *false dismissals*. In particular, each sequence (or *sliding window* in subsequences which consists of a series of values at consecutive timestamps) in the database is first transformed to a lower dimensional point, using any dimensionality reduction technique; then the reduced data are inserted into a multidimensional index (e.g., R-tree [17]) on which *range queries* can be issued to obtain a candidate set; finally, the retrieved candidates are refined by checking their real distances to the query series.

Dimensionality reduction. Since the dimensionality of time series are usually high (e.g., 1,024), the similarity search over high-dimensional index usually encounters a serious problem, known as the "curse of dimensionality." That is, the query performance of the similarity search over indexes degrades dramatically with the increasing dimensionality. In order to break such curse, various dimensionality reduction techniques have been proposed to reduce

the dimensionality of time series before indexing them, including *Singular Value Decomposition* (SVD) [20], *Discrete Fourier Transform* (DFT) [1], [15], DWT [9], *Piecewise Aggregate Approximation* (PAA) [33], *Adaptive Piecewise Constant Approximation* (APCA) [19], *Chebyshev Polynomials* (CP) [8], and *Piecewise Linear Approximation* (PLA) [11]. These reduction methods obey the *lower bounding lemma* [15] such that no false dismissals are introduced during the similarity search. Note that only DFT and DWT have been used in the scenario of stream time series, however, with the limitation that only one pattern is considered. Apart from the dimensionality reduction, VA-file or VA^{+}-file [30] is proposed to improve the search efficiency, which constructs a space-efficient structure for time series and performs a quick *linear scan* on the structure.

Similarity measures. In addition to *Euclidean distance* (L_2-norm) [1], [15], several other distance functions have been proposed to measure the similarity between two time series in different applications such as *Dynamic Time Warping* (DTW) [4], [34], [19], *Longest Common Subsequence* (LCSS) [6], [29], and *Edit Distance with Real Penalty* (ERP) [10]. Specifically, *Euclidean distance* requires the time series to have the same length, which may restrict its applications. On the other hand, DTW can handle sequences with different lengths and *local time shifting*; however, it does not follow the *triangle inequality*, which is one of the most important properties of a *metric* distance function. A recent work [19] makes DTW indexable by approximating time series with bounding envelopes. The resulting R-tree index [17] is clearly inefficient for query processing on stream time series, in terms of both update and search cost. ERP can support *local time shifting* and is a *metric* distance function. LCSS is proposed to handle noise in data; however, it ignores various gaps in between similar subsequences, which leads to inaccuracy. Furthermore, Megalooikonomo et al. [38] proposed *Multiresolution Vector Quantized* (MVQ) representation of time series, which encodes the time series using symbols with their proposed hierarchical distance function. In other applications such as tag visualization [14], hierarchical structure is also used, however, with different semantic meanings (i.e., the most interesting objects within each segment in the structure) from that in the time-series domain. In contrast, our work focuses on L_p-norm, which covers a wide range of applications [33]. Formally, the L_p-norm distance between two series $X(X[0], X[1], \ldots, X[n-1])$ and $Y(Y[0], Y[1], \ldots, Y[n-1])$ of length n is defined as

$$L_p(X,Y) = \sqrt[p]{\sum_{i=0}^{n-1} |X[i] - Y[i]|^p}, \quad (1)$$

where $p \geq 1$. Note that L_1-norm is also called *Manhattan distance*, whereas L_2-norm is *Euclidean distance*. When p is infinite, L_∞-norm is defined as

$$L_\infty(X,Y) = \max_{i=0}^{n-1}\{|X[i] - Y[i]|\}. \quad (2)$$

2.2 Monitoring in Stream Time Series

Not much previous work has been published for monitoring stream time-series data. Zhu and Shasha [37] proposed a method to monitor the correlation among any pair of stream time series within a *sliding window*, in which DFT was used as a summary of the data. Later, they introduced a *shift wavelet tree* (SWT) based on DWT to monitor bursts over stream time-series data [36]. Bulut and Singh [7] improved the technique by using multiscale DWT trees to represent data. Gao and Wang [16] proposed a prediction model to save the computational cost during the matching between a *single stream* time series and *multiple static* patterns. Recently, Papadimitriou et al. [27] proposed a method for capturing correlations among multiple stream time-series data with the help of an incremental PCA computation method. With respect to data estimation, Yi et al. [35] estimated the current values of a co-evolving time series through a multivariate linear regression. These approaches, however, are different from our similarity match problem, in the sense that they do not assume any patterns available. Instead, they aim at detecting either patterns in a stream time series or changes in correlation pattern over multiple stream time series. Moreover, Wu et al. [31] proposed an online matching algorithm for detecting subsequences of financial data over a dynamic database. However, their segmentation and pruning methods are designed for financial data only and cannot be applied to detect general patterns in stream time series.

3 Problem Statement

A *stream time series* is an ordered sequence,

$$S = (s_1, s_2, \ldots, s_i, \ldots, s_t),$$

where each s_i is a real value arriving at a specific time i, and index t is the current timestamp. In this paper, we focus on the *sliding window* model due to its popularity and generality. A *sliding window* is denoted as $W_i = (s_i, s_{i+1}, \ldots, s_{i+w-1})$, where $i = 1, \ldots, (t-w+1)$ and w is the predefined window size.

Assume we have a set of predefined time-series patterns, $P = \{p_1, p_2, \ldots, p_m\}$, where the length of each pattern p_j is equal to w. We formally define the pattern matching problem [22] over stream time series below. Specifically, we want to retrieve the similar match between every window W_i, for $i = 1, \ldots, (t-w+1)$, in stream time series S and patterns $p_j \in P$. Here, a sliding window W_i is similar to a pattern p_j if $dist(W_i, p_j) \leq \varepsilon$, where $dist$ is an L_p-norm distance function ($p \geq 1$) defined in (1) and (2). Moreover, ε is specified by users depending on different applications (or the length, w, of patterns).

Several remarks are given below. First, in stream time series, new data may arrive at a high speed, and therefore we need to process queries very fast. Further, in case a number of new sliding windows from the same stream series arrive and need to be processed, we should design efficient methods to batch process them. This scenario may occur in many applications, such as the delayed sensor data mentioned before. Second, although we define our problem over multiple stream time series S, the problem of detecting *multiple* patterns over *multiple* streams can be essentially induced to that over a *single* stream. Third, we consider a pattern set P containing *multiple* patterns (instead of single

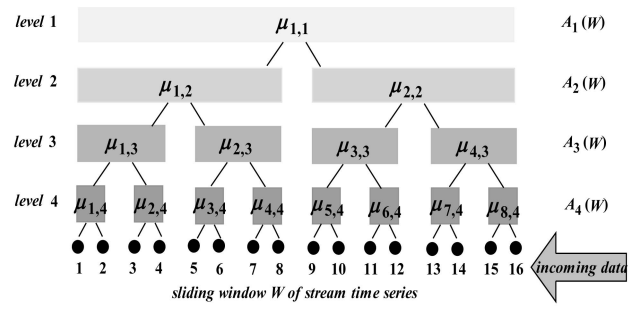


Fig. 1. Illustration of the MSM approximation.

one). Moreover, we first assume patterns are *static*. However, due to the efficient index for patterns, our approach can be easily generalized to the *dynamic* case where patterns are also from *stream time series*. The dynamic property of patterns is in favor of practical applications, which monitor interesting patterns that are updated by either inserting new ones or deleting expired ones. We would discuss this dynamic case later in this paper.

Given a set of predefined patterns, one of the important issues in finding similar patterns over a stream time series is to reduce the processing cost during the detection. The time complexity of directly computing the L_p-norm distance between each pair of series is $O(w)$, which is costly if we perform the matching for each sliding window of stream time series. There are some possible but infeasible solutions. First, we can build a tree index, such as R-tree [17], over the pattern set P. For each incoming W_i, we can conduct the similarity search over the index using W_i as query point. However, due to the "curse of dimensionality" [30] (i.e., costly search in the tree), it is not suitable for the stream processing. Second, another possible approach is to reduce the dimensionality of time series W_i and patterns p_j and search over the reduced data. That is, we reduce W_i and p_j to lower dimensional points, and check the distance between these two points, as a filter, with the guarantee of no false dismissals. However, since the straightforward way is to reduce the dimensionality of each W_i from scratch, the reduction incurs high cost. Motivated by this, in the sequel, we propose a novel multiscale approximation of time series that is efficient for incremental maintenance, followed by a multistep filtering technique to reduce the total cost of pattern matching via the approximation, which are designed for fast pattern matching in stream time series.

4 Multiscale Segment-Mean Approximation

In this section, we propose a novel *MSM approximation* of time series, which is based on the *segment mean* [33]. Specifically, suppose we have a time series W of length w, where $w = 2^l$ for a nonnegative l (note: here we assume the length w of W is a multiple of 2; otherwise, a sequence of 0 can be appended [19], [33]). We construct MSM approximation of W by computing the *segment mean* representations from (the coarsest) level 1 to (the finest) level l. In particular, on each level j, there are totally 2^{j-1} disjoint segments of equal size 2^{l-j+1}, each of which is represented by the mean of all values within itself. Fig. 1 illustrates a simple example of MSM representation of a time series W with length $w = 16$ and thus $l = 4$. On the approximation level 4, there are in total eight segments, each of which contains the mean of two values within itself. For example, the first segment $\mu_{1,4}$ on level 4 is the mean of the first and second values in stream time series W. Similarly, on level 3, there are four segments, each of which is given by the mean of two adjacent segments on level 4 (e.g., $\mu_{1,3} = (\mu_{1,4} + \mu_{2,4})/2$).

Formally, the MSM approximation $A(W)$ for W is denoted as

$$A(W) = [A_1(W), A_2(W), \ldots, A_l(W)], \tag{3}$$

where $A_j(W)$ is a *segment mean* approximation for W on level j for $1 \le j \le l$, and

$$A_j(W) = \left[\mu_{1,j}, \mu_{2,j}, \ldots, \mu_2^{j-1}, j\right], \tag{4}$$

where $\mu_{i,j}$ is the mean of values $m_{i,j}$ within the ith segment in W on level j, for $1 \le i \le 2^j$.

Here, mean is not an additive function. However, since the number of values in a segment on level j is predetermined as 2^{l-j+1}, we can incrementally maintain *sum* in a segment, $(\sigma_{i,j} = \sum_{sk \in mi,j}(s_k))$, for $m_{i,j}$, and efficiently compute the mean, $\mu_{i,j}$, *as* $\sigma_{i,j}/2^{l-j+1}$, when needed. Furthermore, the mean on level j can be computed from the mean on level $j+1$, and treated as an additive function.

Note that the *segment mean* representation of a time series is well studied in literature [33] in the context of static time-series databases. However, few previous works use it with a *multiscale* representation to approximate the time series in the stream scenario. We will show later that this multiscale representation would enable our multistep filtering which can achieve low computational cost during the pattern matching in stream time series. To the best of our knowledge, in the context of time series that contain numerical values, no previous work proposed MSM and the idea of using MSM to perform the multistep filtering. In the sequel, we first give the property of MSM.

Theorem 4.1. *Given two time series of length* $w\ (= 2^l)$, W *and* W', *it holds that*

$$2^{1/p} \cdot L_p(A_i(W), A_i(W')) \le L_p(A_{i+1}(W), A_{i+1}(W')), \tag{5}$$

where $1 \le i < l$.

Proof. Let $f(\cdot)$ be a convex function satisfying

$$f(\lambda_1 x_1 + \cdots + \lambda_b x_b) \le \lambda_1 f(x_1) + \cdots + \lambda_b f(x_b), \tag{6}$$

where x_i are real values, and λ_i are nonnegative real values such that $\sum_{i=1}^{b} \lambda_i = 1$. Based on (6), Yi and Faloutsos [33] showed that for a sequence $W = [s_1, \ldots, s_b]$ and any $p \ge 1$

$$b \cdot |mean(W)|p \le \sum_{i=1}^{b} |s_i|^p \tag{7}$$

holds, where $mean(W)$ is the mean of all values $s_i \in W$. Briefly, let $f(x) = |x|^p$, and $\lambda_i = 1/w$. Then, $f(\sum_{i=1}^{b} \lambda_i s_i) = f(\sum_{i=1}^{b} s_i/w) = |mean(W)|^p$.

By applying (6) and (7) to segments on any two consecutive levels (e.g., jth and $(j+1)$th levels) of times series, we can give our proof of Theorem 4.1 for L_p-norm as follows: by induction.

Base-Case. When $l = 2$, we show that $2 \cdot L_p(A_1(W), A_1(W')) \le L_p(A_2(W), A_2(W'))$ holds.

Here, based on the definition, we have

$$A_1(W) = \mu_{1,1} = \frac{\mu'_{1,2} + \mu'_{2,2}}{2}, \tag{8}$$

$$A_1(W') = \mu'_{1,1} = \frac{\mu'_{1,2} + \mu'_{2,2}}{2}. \tag{9}$$

Applying (7), we have

$$2^{1/p} \cdot |A_1(W) - A_1(W')| \sqrt[p]{\sum_{i=1}^{2} \left|\mu_{i,2} - \mu'_{i,2}\right|^p} \tag{10}$$

which exactly corresponds to (5).

General-Case. We show Theorem 4.1 holds for any level j in a totally l-level approximation. Equation (7) holds for $i = 1$ when $l = 2$ (two levels). Based on this, it holds for every segment $m_{i,j}$ for $1 \le j < l$, because every segment $m_{i,j}$, on level j, consists of two segments on level $j+1$:

$$\mu_{i,j} = \frac{1}{2}(\mu_{2i-1,j+1} + \mu_{2i,j+1}) \quad \text{and}$$
$$\mu'_{i,j} = \frac{1}{2}\left(\mu'_{2i-1,j+1} + \mu'_{2i,j+1}\right).$$

That is, (7) holds for *every* segment $m_{i,j}$ on level j. Given an lth level approximation, for any level j, for $1 \le j < l$, we have

$$\begin{aligned}
&2^{1/p} \cdot L_p\big(A_j(W), A_j(W')\big) \\
&= 2^{1/p} \cdot \sqrt[p]{\sum_{i=1}^{2^{j-1}} \left|\mu_{i,j} - \mu'_{i,j}\right|^p} \\
&\le \sqrt[p]{\sum_{i=1}^{2^j} \left(\left|\mu_{2i-1,j+1} - \mu'_{2i-1,j+1}\right|^p + \left|\mu_{2i,j+1} - \mu'_{2i,j+1}\right|^p\right)} \\
&= L_p\big(A_{j+1}(W), A_{j+1}(W')\big).
\end{aligned}$$

Therefore, (5) holds for any level j. □

Theorem 4.1 provides the relationship of L_p-norm distances between two MSM approximations on levels i and $(i+1)$, respectively. We will discuss later its usefulness for our multistep filtering.

Since the time series $W = [s_1, s_2, \ldots, s_w]$ itself, for $w = 2^l$, can be considered as a special approximation on level $\log_2(w) + 1$, we have the following corollary.

Corollary 4.1. *Given two time series W and W' of length w $(= 2^l)$, we have*

$$2^{\frac{\log_2(w)+1-j}{p}} L_p\big(A_j(W), A_j(W')\big) \le L_p(W, W'). \tag{11}$$

Corollary 4.1 provides a nice lower bound of the L_p-norm distance between time series W and W' (i.e., LHS of Inequality (11)), which is defined by the jth level MSM approximations $A_j(W)$ and $A_j(W')$. This lower bound distance can help prune the search space of pattern matching as follows: Assume W is a query time series. Given a similarity threshold ε, for any j, as long as the lower bound distance defined by $A_j(W)$ and $A_j(W')$ is greater than ε, then the real distance $L_p(W, W')$ between W and W' must be greater than ε (via inequality transition), indicating that W' can be safely pruned. Note that this pruning method (i.e., using the lower bound distance) can guarantee that no false dismissals are introduced.

In the sequel, we briefly compare the MSM approximation with another multiscale representation, DWT [9], which has been widely used for the image retrieval [26] as well as a dimensionality reduction technique [9]. In particular, DWT can also transform the time series data to a multiscale representation, where lower frequency bands are represented as lower levels and higher frequency bands are recorded as higher levels. One of the most popular wavelets is the *Haar wavelet*, which has been widely used in image [28], speech [4], and signal processing [3], since it can be quickly computed with a linear cost. Chan and Fu [9] proved that the L_2-norm distance between the first few *Haar wavelet* coefficients is the lower bound of that between the original time series.

We can obtain a multiscale representation of a time series (with length w) by taking coefficients of the transformed *Haar wavelet* on each level [9]. In particular, for each level i, we take the first 2^{i-1} coefficients of the *wavelet*, where $1 \le i \le \log_2 w$. From [9], we have the following corollary, showing that the relationship of L_2-norm distances between two wavelet representations on levels i and j, respectively, where $i \le j$.

Corollary 4.2. *Given two levels i and j of wavelet coefficients $(i \le j)$ for two time series W and W', the L_2-norm distance computed on level i between W and W' is a lower bound of that computed on level j.*

In the theorem below, we compare the pruning power between MSM and DWT under L_2-norm.

Theorem 4.2. *Given a time series of $W = (s_1, s_2, \ldots, s_w)$ of length $w = 2^l$, let its multiscale wavelet approximation be $H(W) = [h_1, h_2, \ldots, h_w]$, where $h_j = [h_{1,j}, h_{2,j}, \ldots, h_{2^{j-1},j}]$ is the Haar wavelet coefficients on level j [9], and its MSM be $A(W) = [\mu_1, \mu_2, \ldots, \mu_w]$ with jth level approximation $\mu_j = [\mu_{1,j}, \mu_{2,j}, \ldots, \mu_{2^{j-1},j}]$. For any level $1 \le j \le l$, we have $|h_j|^2 = 2^{l+1-j}|\mu_j|^2$, where $|x|$ is the euclidean norm of vector x.*

Proof. Please refer to the Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2008.184. □

From Theorem 4.2, it is worth noting that, the pruning power of MSM is the same as that of DWT under L_2-norm. However, since DWT is an orthogonal linear transformation, only L_2-norm is preserved under this transformation [33]. In other words, for L_p-norm other than L_2-norm, the distance of DWT is *not* preserved at all. Therefore, compared to our proposed MSM that can work under arbitrary L_p-norm, DWT is only limited to L_2-norm. Furthermore, since for an l-scale approximation, MSM needs to compute 2^l means for segments, whereas DWT has to compute 2^l more *wavelet coefficients* in addition to means. Thus, the computational cost of DWT is higher than that of our MSM approximation for the similarity search under L_p-norm.

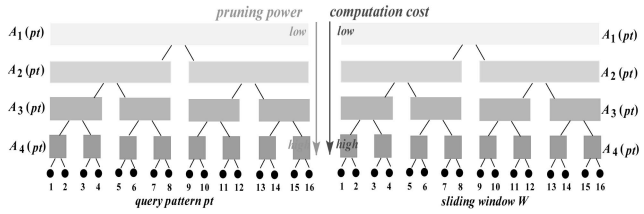


Fig. 2. Pruning heuristics of multistep filtering.

5 Multistep Filtering Method

Section 5.1 illustrates the rationale of our pruning technique. Section 5.2 proposes the algorithm for segment mean pruning (SMP). Section 5.3 gives other pruning schemes.

5.1 Pruning Heuristics

As indicated by (11), MSM approximations of two time series W and W' on *any* level j can define a lower bound of their real distance, which can be used to fast pattern matching in stream time series. However, different levels of approximation have different computational cost and pruning power. As illustrated in Fig. 2, we have two time series pt (pattern) and W. In order to calculate the lower bound distance using the first level MSM, we only need $O(1)$ computational cost, involving one value for each MSM; on level 2, the computational cost is doubled, compared with level 1, with two values involved each; and so on. On the other hand, as indicated by (5) in Theorem 4.1, the lower bound distance on level i is never greater than that on level j, for $i < j$, which shows that high-level MSM would result in high pruning power (due to the tight lower bound distance). In turn, high pruning power produces fewer candidates, whose refinement by computing their real distances is much smaller.

Thus, it motivates us to find a strategy to perform the pruning based on MSM approximations. In particular, we need to decide which level(s) of MSM we should use for the pattern matching, making a balance between computational cost and pruning power. In the sequel, Section 5.2 proposes a novel *multistep filtering* method on MSM to perform the pattern matching, which aims to reduce the total cost. Section 5.3 discusses other possible pruning schemes.

5.2 Segment Mean Pruning

Next, we present a simple yet effective pruning algorithm, namely SMP to check whether or not a sliding window W of size $w\ (= 2^l)$ matches with any pattern pt in a pattern set P. Specifically, since the size of the pattern set P can be large, it is not efficient to compare every pattern pt in P with W during the matching process. Thus, we propose a d-dimensional grid index GI to store patterns, where $d\ (= 2^{lmin-1})$ is typically low (e.g., 1 or 2) to provide low space and accessing costs. In particular, we split the data space into cells with equal-sized bins along each dimension. For each pattern $pt \in P$, we compute its l_{min}th level MSM approximation $A_{lmin}(pt)$ and store pt (with pattern identifiers and MSM approximations) in a cell of GI that contains $A_{lmin}(pt)$. Then, during the stream processing (SMP), given a sliding window W, we can efficiently access those patterns in cells of GI that are close to $A_{lmin}(W)$. As an example in Fig. 2, assuming $l_{min} = 1$ (i.e., $d = 1$), for the query pattern pt of length 16 in P, we start with level 4 (i.e., $A_4(pt)$) and incrementally compute MSM approximations level by level until level 1 (i.e., $A_1(pt)$). Then, we insert $A_1(pt)$ into a 1D grid GI, where pattern pt in GI falls into a cell that contain $A_1(pt)$, and the size of each cell is set to ε. Similarly, when $d = 2$ (i.e., $l_{min} = 2$), we construct a 2D grid GI containing cells with size $\varepsilon/\sqrt{2}$ and store the second-level MSM approximation $A_2(pt)$ of pattern pt in GI. Note that, although we consider the grid of equal size here, it can be easily extended to that of skewed sizes, adaptive to the distribution of patterns. Here, the time cost of offline constructing a grid index is given by $|P| \cdot \sum_{i=l_{\min}-1}^{l-1} 2^i$ (i.e., $O(|P| \cdot 2^l)$), where $|P|$ is the total number of patterns and 2^l is the length of pattern.

Algorithm 1. *SMP*
Input: a pattern set P, a sliding window W of length $w\ (= 2^l)$, a similarity threshold ε, and a d-dimensional grid GI on P
Output: a candidate pattern set P' matching with W.

1. let T contain all ids for patterns in P that may match with W through the grid GI
2. $j = l_{min} + 1$
3. **while** $(j \le l)$ **and** $(T \neq \phi)$ **and** E_j **do** // *E_j is the early stop condition*
4. $T' \leftarrow \phi$
5. **for** each $pt \in T$ **do**
6. **if** $L_p(A_j(pt), A_j(W)) \le \varepsilon/2^{l+1-j}$ **then** // *SS pruning scheme*
7. $T' \leftarrow T' \cup \{pt\}$ // *candidate patterns to be pruned in next level*
8. **end if**
9. **end for**
10. $T \leftarrow T'$
11. $j = j + 1$
12. **end while**
13. let P' contain all the patterns with ids in T'
14. **return** P'

Algorithm 1 illustrates the details of SMP, which performs the pattern matching when a new sliding window W from a stream time series is obtained. Basically, we first retrieve those candidate patterns in cells of GI that are within ε distances from $A_{lmin}(W)$, and then filter out false alarms by MSMs on different levels. Given a pattern set P, a similarity threshold ε, and a d-dimensional grid index GI built on P, SMP returns a candidate pattern set P' (subset of P) in which patterns cannot be pruned by our pruning method (i.e., they may match with the sliding window W).

Without loss of generality, assume that we have precalculated MSM approximations $A_j(pt)$ for all the patterns $pt \in P$ on levels $j\ (l_{min} < j \le l)$, and store them in grid index GI. Each step of Algorithm 1 is given as follows: First, we access grid index GI to obtain all the pattern identifiers in the cells that have their minimum distances from $A_{lmin}(W)$ smaller than or equal to ε (line 1). Then, we initialize a set T containing identifiers of the result (line 1) and set j to $l_{min} + 1$ (line 2). SMP prunes patterns in P for sliding window W in a while-statement using MSMs from level $l_{min} + 1$ up to l at most (lines 3-12). As mentioned in Section 5.1, the distance computation with higher level MSM approximations is more costly, however, achieving more pruning power. Thus, we need an early

stop condition, denoted as E_j, to decide up to which level we should terminate the pruning procedure (i.e., while-statement) such that the total cost can be minimized. We will discuss the detailed definition of early stop condition E_j later in this section. In our algorithm, the while loop terminates, when any of the three conditions holds below: 1) j reaches the last approximation level l, 2) there are no matching patterns left, or 3) the early stop condition, E_j, is satisfied. Within each iteration j, there is a for-statement to check whether or not each pattern pt in T satisfies (11) (line 6). If it is met, pt is added to T', which is the candidate pattern to be pruned in the next level (line 7). Finally, we return a candidate pattern set P' with identifiers in T' that may match with W (lines 13 and 14).

We now give the early stop condition E_j to terminate the pruning procedure. Note that there are totally $|P|$ patterns. Let P_j be the probability that sliding window W cannot be pruned using the j-level MSM approximation, which can be collected from statistics during the query processing. Since using higher level of MSM can achieve greater pruning power, it is expected to hold that $P_i \geq P_j$, for $i < j$. Assuming the average cost of computing distance between two series along each dimension is C_d, the total computational cost, $cost_j$, that SMP stops pruning up to level j ($\leq l$) and computes the real distance with unpruned candidates is given by

$$\cos t_j = \sum_{i=l_{\min}}^{j-1} \left(P_i \cdot |P| \cdot 2^i \cdot C_d\right) + P_j \cdot |P| \cdot w \cdot C_d. \quad (12)$$

In (12), the first term is the total cost of our pruning from level l_{min} to level j. As an example, with probability P_{j-1}, sliding window W cannot be pruned by the $(j-1)$th level, and we need to compute distance for each object from $|P|$ patterns using 2^{j-1} segments on level j (i.e., with $2^{j-1} \cdot C_d$ cost). Therefore, the computational cost on level $j-1$ is expected to be $P_{j-1} \cdot |P| \cdot 2^{j-1} \cdot C_d$. The second term in (12) is the expected cost to compute the actual distances between $|P|$ patterns and the remaining sliding windows W (if W cannot be pruned by jth level MSM approximation). It is worth noting that we can use level j to do the further filtering only if

$$cost_{j-1} \geq cost_j. \quad (13)$$

Furthermore, based on (12) and (13), we have

$$\log \frac{P_{j-1} - P_j}{P_{j-1}} \geq (j - 1 - \log(w)). \quad (14)$$

Specifically, if (14) holds, we can continue filtering using level j; otherwise, we stop checking the next level. Thus, (14) exactly corresponds to the early stop condition E_j in Algorithm 1. Note that, here P_j can be obtained during the query processing, which is the number of sliding windows that cannot be pruned by jth level MSM divided by the total number of sliding windows. In practice, in order to speed up the procedure, we can take a few samples of sliding windows among all the sliding windows to estimate P_j.

As evaluated by our experiments, j is usually much smaller than l ($\log w$), and therefore the filtering scheme is efficient by comparing only a few levels of approximations.

5.3 Other Pruning Schemes

Since the pruning approach described in Algorithm 1 prunes patterns with MSM approximations level by level, we call it simply as SS scheme. In addition, we also considered two other schemes, including *jump-step* (JS) and *one-step* (OS) schemes. Specifically, JS scheme uses only two levels of MSM to prune, that is, first choosing level $(l_{min} + 1)$ to prune and then jumping to the jth level. On the other hand, OS scheme uses only one level, that is, the jth level for pruning. As we discuss further on, SS is the best one among the three as long as certain conditions hold. In the sequel, we denote cost $cost_j$ in (12) of SS as $cost_{\mathrm{SS}}$.

First, we consider JS scheme which only selects two levels $(l_{min} + 1)$ and j to prune patterns. Similar to SS scheme, the total computational cost, $cost_{\mathrm{JS}}$, of JS is given as follows:

$$cost_{JS} = P_{l_{\min}} \cdot |P| \cdot 2^{l_{\min}} \cdot C_d + P_{l_{\min}+1} \cdot |P| \cdot 2^{j-1} \cdot C_d + P_j \cdot |P| \cdot w \cdot C_d. \quad (15)$$

In (15), the first term is the cost of pruning using the l_{min}th level MSM, the second term is that of pruning via the jth level MSM, and the last term is that of computing real distances between patterns and the remaining sliding windows.

In the following theorem, we give the condition that SS can outperform JS (i.e., $cost_{\mathrm{SS}} \leq cost_{\mathrm{JS}}$).

Theorem 5.1. *Let $cost_{\mathrm{SS}}$ and $cost_{\mathrm{JS}}$ be the costs of SS and JS schemes, respectively. If it holds that $P_{lmin+1} \geq 2 \cdot P_{lmin+2}$, then we have $cost_{\mathrm{SS}} \leq cost_{\mathrm{JS}}$, where P_j is the percentage of sliding windows that cannot be pruned via jth level MSM, and l_{min} is the lowest level of MSM we compute.*

Proof. Based on (15) and (12), we have

$$\begin{aligned} & cost_{\mathrm{SS}} - ost_{\mathrm{JS}} \\ &= |P| \cdot C_d \cdot \left(\left(\sum_{i=l_{\min}}^{j-1} (P_i \cdot 2^i) + P_j \cdot w \right) \right. \\ &\qquad \left. - \left(P_{l_{\min}} \cdot 2^{l_{\min}} + P_{l_{\min}+1} \cdot 2^{j-1} + P_j \cdot w\right) \right) \quad (16) \\ &= |P| \cdot C_d \cdot \left(\sum_{i=l_{\min}+1}^{j-1} (P_i \cdot 2^i) - P_{l_{\min}+1} \cdot 2^{j-1} \right). \end{aligned}$$

Since $P_{lmin+2} \geq P_i$ $(i \geq l_{min} + 2)$, we replace all such P_i $(i \geq l_{min} + 2)$ with P_{lmin+2} in (16), thus having

$$\begin{aligned} & cost_{\mathrm{SS}} - cost_{\mathrm{JS}} \\ &\leq |P| \cdot C_d \cdot \left(P_{l_{\min}+1} \cdot \left(2^{l_{\min}+1} - 2^{j-1}\right) + P_{l_{\min}+2} \cdot \sum_{i=l_{\min}+2}^{j-1} 2^i \right) \\ &= |P| \cdot C_d \cdot \left(P_{l_{\min}+1} \cdot \left(2^{l_{\min}+1} - 2^{j-1}\right) - P_{l_{\min}+2} \cdot \left(2^{l_{\min}+2} - 2^j\right)\right). \quad (17) \end{aligned}$$

Note that, if the right-hand side of (17) is nonpositive, then we have

$$|P| \cdot C_d \cdot \left(P_{l_{\min}+1} \cdot (2^{l_{\min}+1} - 2^{j-1}) - P_{l_{\min}+2} \cdot (2^{l_{\min}+2} - 2^j)\right) \leq 0, \quad (18)$$

or equivalently we can obtain $P_{lmin+1} \geq 2 \cdot P_{lmin+2}$. In other words, when it holds that $P_{lmin+1} \geq 2 \cdot P_{lmin+2}$, we have $cost_{\mathrm{SS}} - cost_{\mathrm{JS}} \leq 0$, which completes the proof. □

Next, we discuss OS scheme which uses only one level j for pruning. Similarly, the computational cost $cost_{\mathrm{OS}}$ of OS is given by

$$\cos t_{\mathrm{OS}} = P_{l_{\min}} \cdot |P| \cdot 2^{j-1} \cdot C_d + P_j \cdot |P| \cdot w \cdot C_d. \quad (19)$$

In (19), the first term is the cost of pruning via the jth level MSM, and the second term is that of computing real distances between patterns and the remaining sliding windows.

Similar to JS, we give the condition that SS outperforms OS in the theorem below (i.e., $cost_{\mathrm{SS}} \leq cost_{\mathrm{OS}}$).

Theorem 5.2. *Let $cost_{\mathrm{SS}}$ and $cost_{\mathrm{OS}}$ be the costs of SS and OS schemes, respectively. If it holds that $P_{lmin} \geq 2 \cdot P_{lmin+1}$, then we have $cost_{\mathrm{SS}} \leq cost_{\mathrm{OS}}$, where P_j is the percentage of sliding windows that cannot be pruned via jth level MSM, and l_{min} is the lowest level of MSM we compute.*

Proof. Refer to Appendix B, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2008.184. □

Theorems 5.1 and 5.2 indicate that, on level $(l_{min}+1)$, as long as the approximation can filter out more than 50 percent of the objects, SS has lower cost than OS. Similarly, if the level $(l_{min}+2)$ approximations have the ability to filter out at least 50 percent of the objects left from the level $(l_{min}+1)$ pruning, SS outperforms JS. By investigating extensive experiments, we find that such filtering percentages are above 50 percent, which shows by theorems that we should select SS as our pruning scheme to achieve low cost. Finally, our SS pruning scheme uses approximations up to level l_{max}. Thus, the space requirement of each sliding window W is $2^{l_{max}-1}$. That is, it is enough to store approximation $A_{l_{max}-1}(W)$ on level $(l_{max}-1)$, from which lower levels can be computed.

6 Query Processing

Section 6.1 illustrates the detailed query procedure of pattern matching over stream time series. Section 6.2 discusses the batch processing of pattern matching.

6.1 Similarity Matching

Based on MSM mentioned in Section 3, Algorithm 2 (*Similarity_Match*) illustrates the query procedure to retrieve the pattern matches between patterns and a stream time series S. In particular, when a new data item *new_d* arrives at S, we obtain the latest sliding window W_i that contains this new data (lines 1 and 2). Then, we invoke Algorithm 1, **SMP**, to find all the candidate patterns (i.e., P') that may potentially match with W_i (line 3). Then, for each candidate pattern $pt \in P'$, we calculate the real distance from pt to W_i, and finally output the pair (W_i, pt) if they are similar (lines 4-8).

Algorithm 2 ***Similarity_Match***
Input: a time-series stream S, a set P of patterns and ε, and grid index GI on P
Output: a set of matching pairs

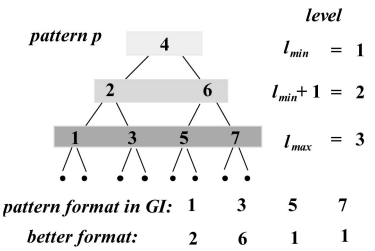


Fig. 3. Example of pattern representation.

1. **while** a new data item *new_d* arrives **do**
2. let W_i be a sliding window with the w most recent values containing *new_d*
3. $P' \leftarrow$ **SMP** $(P, W_i, \varepsilon, GI)$
4. **for** each $pt \in P'$ **do**
5. **if** $\mathrm{L}_p(W_i, pt) \leq \varepsilon$, **then**
6. **output** the similar pair (W_i, pt)
7. **end if**
8. **end for**
9. **end while**

Note that, here each pattern pt in the grid index GI is represented by 2^{lmax-1} values on level l_{max} in the form $A_{lmax}(pt)$, where l_{max} is the highest level MSM we store in GI for patterns (besides the actual data). As an example in Fig. 3, a pattern pt is with an MSM representation from level l_{min} to l_{max}, assuming $l_{min} = 1$ and $l_{max} = 3$. In the grid index GI, we store the level 3 approximation $A_3(pt)$ (i.e., on level l_{max}), since both $A_1(pt)$ and $A_2(pt)$ on levels 1 and 2, respectively, can be calculated from $A_3(pt)$. That is, $A_2(pt) = <(1+3)/2, (5+7)/2>$ and $A_1(pt) = <(1+3+5+7)/4>$.

However, since the procedure applies SS scheme (line 3 in Algorithm 2), we need to perform the matching test with approximations of pt from level $(l_{min}+1)$ to l_{max}, where each level has to be calculated from $A_{lmax}(pt)$. This incurs high computational cost, and thus optimizations need to be carried out. In particular, for any pattern pt, we record its level $(l_{min}+1)$ approximation $A_{lmin+1}(pt)$ in GI. In addition, similar to DWT, we also store differences of level $(l_{min}+2)$ from level $(l_{min}+1)$, level $(l_{min}+3)$ from level $(l_{min}+2)$, and so on. As in the example, we keep the pattern in the form $<2, 6, 1, 1>$ in GI, where the first two values "2" and "6" are on level 2 $(= l_{min}+1)$, the third "1" is the difference between "3" and "2" on levels 3 and 2, respectively, and the second "1" is that between "7" and "6" also on levels 3 and 2, respectively. The advantage of this representation is that, approximation of patterns on the next level can be always calculated with low cost using differences, and computational cost can be saved if SS scheme aborts early. In summary, the space requirement of grid GI is given by $2^{lmax-1} \cdot |P|$, where l_{max} is the maximum possible level used in SS, and $|P|$ the number of patterns.

6.2 Batch Processing Optimization

In Algorithm 2, we consider only one sliding window W_i from a time-series stream arriving at a time. In some scenarios (e.g., in the application with the delayed sensor data), however, a batch of data may arrive at the same time. In addition, in other applications, data are buffered in a

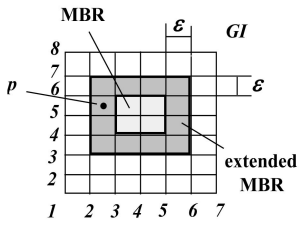


Fig. 4. Pattern matching in grid index GI with extended MBR.

large pool and they are not processed until enough data are collected. If we still apply Algorithm 1 for each sliding window at a time, the cost of looking up the grid index GI would become distractively high. For example, with 1,000 sliding windows, we have to loop up grid 1,000 times. Motivated by this, in the sequel, our work aims to reduce such cost by looking up GI for a group of sliding windows (instead of one) at a time. In the previous example, 1,000 windows can be assigned to several, say 5, groups, thus, we only need to access grid five times, avoiding much cost of accessing the index (i.e., computing cells involved and retrieving candidates). Given a stream time series S, we have a number of consecutive sliding windows, for example, $\{W_i, W_{i+1}, \ldots, W_{i+m}\}$, which can be viewed as one group to look up GI once. In particular, we can obtain the l_{min}th level MSMs from these sliding windows, that is, $A_{lmin}(W_i), A_{lmin}(W_{i+1}), \ldots, A_{lmin}(W_{i+m})$, respectively, and then compute an extended *minimum bounding rectangle* (MBR). We illustrate the extended MBR using a simple example in Fig. 4. Assume we have a grid index GI, where the cell size is set to $\varepsilon\,(=1)$ for a simple illustration. Moreover, a 2×2 MBR (i.e., [3, 5; 4, 6], the internal light gray region) is a rectangle that tightly bounds the second (i.e., l_{min}th) level MSMs obtained from consecutive sliding windows W_j $(i \le j \le i+m)$. Since we want to retrieve those patterns pt in P whose distance to at least one W_j in MBR is within ε, we expand MBR by extending its length by ε on both sides along each dimension. Thus, the resulting 4×4 rectangle (i.e., [2, 6; 3, 7]) is called extended MBR. Any pattern that falls in the extended MBR is considered as a candidate that might match with the group of sliding windows (that define the extended MBR).

In the sequel, we discuss two critical questions of this method:

- How to perform the pattern matching on the grid index using MBR?
- How to split consecutive sliding windows, say $W_i, W_{i+1}, \ldots,$ and W_{i+m}, into several groups?

Any pattern pt falling into this extended MBR is a candidate that can be used for pruning. Therefore, for each candidate pt in the extended MBR, we apply the SS scheme to compute distances from level $(l_{min}+1)$ between approximations of pt and data (sliding windows W_j) in the extended MBR. As in the previous example, since $l_{min}=2$, we obtain the approximation $A_3(pt)$ on level 3, which is then compared with every data in the MBR using the same level approximation. Then, we prune the data with levels 4, 5, and so on. This procedure repeats until either all data are pruned or level l_{max} is reached. Finally, the actual distance between every pattern and data left is computed against threshold ε, and output if they are similar.

The second question arises regarding how to group consecutive sliding windows. Intuitively, we want to group those sliding windows whose extended MBR is small, resulting in lower cost of looking up GI. More specifically, assume we have already grouped the approximations $A_j(W_1), A_j(W_2), \ldots,$ and $A_j(W_k)$ of k consecutive sliding windows $W_1, W_2, \ldots,$ and W_k, respectively. When the next data item arrives, we obtain a new sliding window $A_j(W_{k+1})$. Our problem is, whether or not we should consider $A_j(W_{k+1})$ as being in the same group as previous k approximations. Note that the inclusion of $A_j(W_{k+1})$ may increase the volume of the MBR, resulting in more candidate patterns being checked against data at a high cost. Furthermore, if we start a new MBR for $A_j(W_{k+1})$ (i.e., the number of grouped MBRs is increased by 1), then we need to access the grid one more time in order to retrieve candidates during the query processing, which would increase the lookup cost.

In order to balance the trade-off between both factors, we propose a method to split m approximations. In particular, we define a measure MEA_k to evaluate the goodness of MBR, based on which we perform the grouping. Recall that we have k data bounded by an MBR. We extend both sides of the MBR by ε along each dimension, and obtain an *extended* one. The measure MEA_k is given by the number of patterns falling into the *extended MBR* (containing k windows) divided by k. Intuitively, if a group contains many sliding windows close to each other, then the number of retrieved candidate patterns is expected to be small, that is, measure MEA_k will be small, which would result in low refinement cost. When we want to include the $(k+1)$th point, if the updated measure $MEA_{k+1} \le MEA_k$, we group the new window to previous k ones; otherwise, restart a new MBR which is initialized with the new point. This way, we can obtain approximations adaptive to the dynamic data and moreover reduce the index access. In practice, the number of patterns within a range can be estimated by a space-efficient histogram built for patterns in GI.

7 Dynamic Patterns

In previous sections, we always assume that the pattern set P is static. Now, we extend it to the *dynamic* case where both sliding windows W_i and patterns P come from a set of stream time series. Specifically, we want to find all pairs of stream time series that have the distance between W_i and P within a matching threshold ε. We use the same grid structure GI as we proposed for static patterns to index stream patterns on level l_{min}. Since the grid structure has linear update cost, patterns can be efficiently maintained in this index. After we obtain all candidate pairs, they are further investigated and handled with two scenarios. In the first scenario, candidate pairs appeared in the matching result at the previous timestamp. In this case, instead of computing their real *Euclidean distances* from scratch, we can obtain their distances incrementally, whose cost is only $O(1)$, saving much computational cost. In contrast, in the second scenario, candidate pairs are absent in the result at the previous timestamp. Therefore, similar to the *multiscale*

filtering with static patterns, we have to use the *multistep filtering*, SS, to prune candidates. After the filtering, the remaining candidates are finally refined by calculating their real distances and output. Here, the space consumption is proportional to the total number of sliding windows and dynamic query patterns.

Note that, in our method, all the resulting real distances between pairs are recorded in order to reduce the number of distance computations for the next timestamp. In particular, we use an array to record these distances. Each entry in the array has a combined key, that is, IDs of pairs, by which we can easily retrieve distances that have been computed before. Thus, in the first scenario, distances can be incrementally computed from the result at the previous timestamp, without calculating them from scratch. For example, given two time series $R = \{r_1, r_2, \ldots, r_w\}$ and $S = \{s_1, s_2, \ldots, s_w\}$ of length w and their $\mathrm{L}_1\text{-norm}(R, S)$, when the new data items r_{w+1} and s_{w+1} arrive at R and S, respectively, the new L_1-norm distance $\mathrm{L}_1\text{-norm}(R', S')$ between sequences, $R' = \{r_2, r_3, \ldots, r_{w+1}\}$ and $S' = \{s_2, s_3, \ldots, s_{w+1}\}$, of the latest sliding window can be computed from the previous result, that is,

$$\mathrm{L}_1\text{-norm}(R', S') = \mathrm{L}_1\text{-norm}(R, S) - |r_1 - s_1| + |r_{w+1} - s_{w+1}|.$$

The computational cost of updating one real distance incrementally is only $O(1)$, which is much lower than that of either the *multistep filtering* or simply calculating distances from scratch. The procedure of the similarity match is similar to Algorithm 2, besides probing the array that stores previously computed distances and incrementally computing distances (lines 5 and 6).

8 Experimental Evaluation

In this section, we report the empirical study on the efficiency and effectiveness of our proposed pruning method. The evaluation consists of two parts. First, we verified that our multistep filtering, SS scheme, with MSM indeed improves the retrieval efficiency, compared to other methods OS and JS. Moreover, we tested analytic results of SS. As a second step, we compared the MSM with DWT representation, in terms of efficiency in detecting patterns. Both real and synthetic data were used in the experiments. Specifically, for the real data, we use 24 *benchmark* data sets (each data set consists of 200 series with length 256), and a stock data set [25] which contains two years' New York Stock Exchange (year 2001 and 2002) tick-by-tick real data (15 data sets, each series with length 512). For the synthetic data, both stream and pattern time series data are generated using the *random walk* model (each time series is of length 1,024). For stream S, an element s_i of S is $s_i = R + \sum_{j=1}^{i} (\mu_j - 0.5)$, where R is a constant real number in [0, 100], and μ_j a set of uniform random numbers in [0, 1]. Since our similarity match over the stream time series happens in memory, we assume that at each timestamp, the last sliding windows of all stream time series can fit in the memory (note: the MSM representation for each sliding window of size w is at most $(w - 1)$). Further, since our pruning method uses the lower bound distance to perform the pruning, no false dismissals are introduced for our pattern matching [33] (mentioned in Section 4). Thus, we measure the query performance in terms of the computation time of pattern matching in stream time series, which can also reveal the pruning power and savings of the computational cost. We conducted experiments on a Pentium 4 PC with 512M memory. All results are averaged over 20 runs.

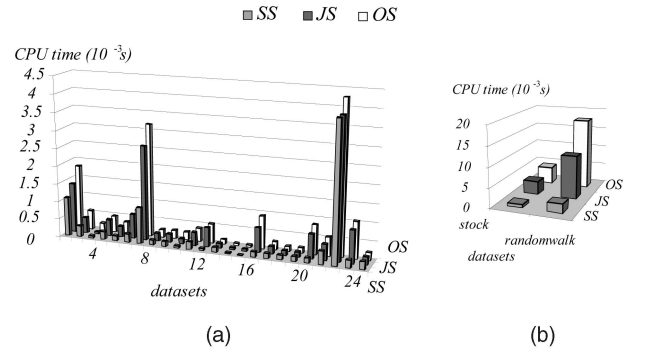


Fig. 5. CPU time comparison of different filtering mechanisms (L_2-norm). (a) Twenty-four *benchmarks*. (b) *Stock* and *randomwalk*.

8.1 Efficiency Test on Multistep Filtering Strategy

In our first experiment, we verified the efficiency of the multistep filtering SS over 24 *benchmark* data sets [16], [34], [7], compared to OS and JS. Since SS, JS, and OS schemes prune patterns using different levels of MSM approximations and via different strategies, in the sequel, we empirically test the execution time of the three methods to see which scheme is the best on real data sets. Here, the *benchmark* data represent a wide spectrum of applications and data characteristics. We randomly picked up a time series from each data set as pattern and conduct a pattern matching query on all levels (i.e., computing distance on each level and checking their distances are within a radius). Then, we compared three different filtering schemes, SS, JS, and OS, using the MSM in L_2-norm (i.e., *euclidean distance*). Fig. 5a shows the experimental result, where the X-axis is 24 *benchmark* data sets sorted in alphabetical order and the Z-axis is the CPU time. As we expected, MSM representation together with the multistep filtering performs efficiently (within a few milliseconds) on all the 24 data sets. Specifically, the filtering mechanism SS achieves the best performance, followed by JS and OS. By further investigating the average pruning power of the first level in an MSM, we found that the first-level representation indeed filtered out over 50 percent of the data in each data set. Again, by checking the average number of remaining results after the filtering on each level, we did find that $P_2 < 50\,\%P_1$ holds. Fig. 5b illustrates the same set of experiments on the real *stock* and synthetic *randomwalk* data sets with similar results. Similar results are obtained under L_p-norm, where $p = 1, 3, \text{and } \infty$.

In the next experiment, we verified the relationship between the highest "useful" level j and $\log \frac{P_{j-1} - P_j}{P_{j-1}}$ based on (16). In particular, we use SS filtering scheme to prune from level 1 to l, and stops on level j, when $(j - 1 - \log w) > \log \frac{P_{j-1} - P_j}{P_{j-1}}$ holds. In order to online obtain P_j, we chose the first 10 percent data and calculated the percentage of samples that

TABLE 1
Test the Analytic Results of Multistep Filtering
(Four Sample Data Sets from 24 Benchmarks)

data-sets	*measure*	*level* (*pattern length* = 256)							
		1	2	3	4	5	6	7	8
	j–1–log w	-8	-7	-6	-5	-4	-3	-2	-1
cstr	$\log \frac{P_{j-1}-P_j}{P_{j-1}}$	**-0.06**	**-0.28**	**-2.14**	**-3.02**	-9.58	-5.34	-9.47	-8.54
	CPU (10^{-4} *s*)	-	1.15	1.55	**1.1**	8	7.5	2.6	8
soil-temp	$\log \frac{P_{j-1}-P_j}{P_{j-1}}$	**-0.19**	**-0.44**	**-0.71**	**-1.52**	**-2.13**	-.3.38	-3.87	-9.62
	CPU (10^{-4} *s*)	-	10.2	10.25	3.95	**1.6**	2.35	2.35	3.95
suns-pot	$\log \frac{P_{j-1}-P_j}{P_{j-1}}$	**-0.13**	**-0.37**	**-0.64**	**-3.27**	**-3.92**	-3.7	-4	-9.6
	CPU (10^{-4} *s*)	-	5.5	1.6	4.7	**1.55**	7.5	3.15	8
ball-beam	$\log \frac{P_{j-1}-P_j}{P_{j-1}}$	**-0.3**	**-0.85**	**-0.79**	**-0.72**	**-1.43**	**-1.6**	-3.51	-6.34
	CPU (10^{-4} *s*)	-	11.75	18	12.4	13.9	**11**	23.5	22

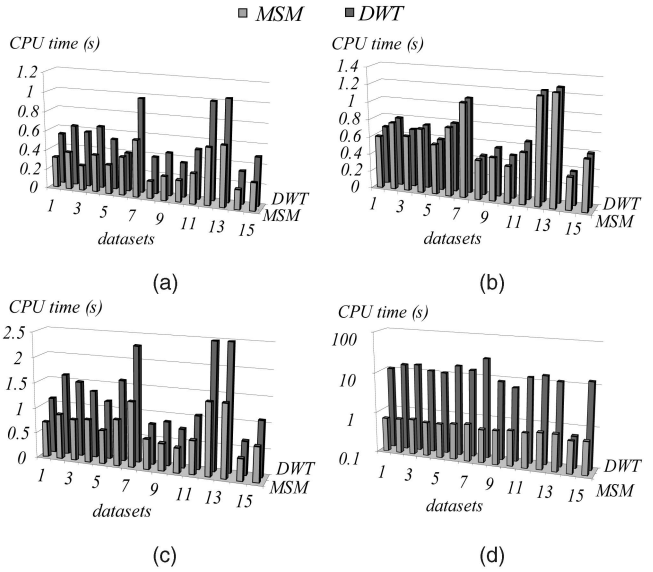


Fig. 6. Comparison between MSM and DWT (15 stock data sets). (a) L_1-norm. (b) L_2-norm. (c) L_3-norm. (d) L_∞-norm.

are left by filtering on level j. Note that, in case we need to continue the filtering on level j, the cost of estimating P_j by computing the distances from 10 percent selection to query pattern (on the jth level of MSM) is in fact included in the cost of filtering. In addition, the time complexity of estimating P_j is only $O(1)$, which is not greater than even one distance computation $O(2^j)$ with the jth level MSM. Table 1 illustrates the CPU time of SS filtering on different levels using four sample *benchmark* data sets, *cstr*, *soiltemp*, *sunspot*, and *ballbeam* (other 20 data sets work as well). Specifically, for each data set, we obtain the value of $\log \frac{P_{j-1}-P_j}{P_{j-1}}$ on different levels. If $\log \frac{P_{j-1}-P_j}{P_{j-1}} \geq (j-1-\log w)$, we highlight the value by bold font. We can see that the maximum level that the bold font is exactly where SS achieves the best performance (i.e., the lowest CPU time).

As illustrated in Table 2, similar results are obtained when we perform the same set of experiments on the *stock* and *randomwalk* data sets. Specifically, patterns in the *stock* data set are of length 512, which contains nine levels, whereas those in *randomwalk* at a length of 1,024, which covers 10 levels in their MSMs. It is easy to see that both data sets have their optimal filtering levels (at which the CPU time is the minimum among all levels) suggested by (16).

TABLE 2
Test the Analytic Results of Multistep Filtering
(Stock and Random Walk Data Sets)

data-set	*measure*	*level* (*pattern length* = 512)								
		1	2	3	4	5	6	7	8	9
	j–1–log w	-9	-8	-7	-6	-5	-4	-3	-2	-1
stock (512K)	$\log \frac{P_{j-1}-P_j}{P_{j-1}}$	**-0.01**	**-0.09**	**-0.47**	**-1.28**	**-2.61**	**-1.47**	-7.52	-3.84	-7.04
	CPU (10^{-3} *s*)	-	63	26	54	54.5	**9**	23.5	31	23
data-set	*measure*	*level* (*pattern length* = 1024)									
		1	2	3	4	5	6	7	8	9	10
	j–1–log w	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
ranwlk (1024K)	$\log \frac{P_{j-1}-P_j}{P_{j-1}}$	**-0.03**	**-0.09**	**-1.75**	-12.9	-12.9	-12.9	-12.9	-12.9	-12.9	-12.9
	CPU (10^{-3} *s*)	-	70	**16**	46	75	75	75	80	80	23.5

8.2 Performance of Detecting Static Patterns

In this section, we report the efficiency test of our MSM representation together with SS filtering in detecting patterns, compared to DWT. Since we consider sliding windows, we perform our experiment on *stock* and *randomwalk* data sets, other than 24 *benchmark* data sets which are only with length 256. For stock data, we randomly choose 1,000 series with length 512 from the generated stock data as patterns, and use the rest as data from stream time series. For *randomwalk*, we randomly obtain 1,000 patterns of length 1,024. Specifically, for the stream time series, we consider a window 1.5 times that the length of patterns. That is, we set the window size to 768 for *stock* data and 1,536 for *randomwalk*. Note that directly computing distances between pairs of pattern and sliding window is very inefficient, for example, the required CPU time under L_2-norm for *sstock* is 1.398 seconds, and that for *randomwalk* is about 4.792 seconds. Thus, we test our approach, which we assume there is a 1D grid index built for 1,000 patterns (i.e., $l_{min} = 1$). Therefore, the filtering step always starts from the second level.

For fair comparisons, we use the same level and number of coefficients in both MSM and DWT. Fig. 6 illustrates the CPU time of MSM and DWT with 15 stock data sets under different L_p-norms. In particular, Figs. 6a, 6b, 6c, and 6d illustrate the experimental results with stock data using L_1-, L_2-, L_3-, and L_∞-norms, respectively. The evaluated CPU time includes two parts, the cost of *incremental* updates and similarity search. For L_2-norm, DWT representation has a similar performance to our MSM. However, since the update cost of wavelet coefficients is higher than that of ours, DWT is slightly worse than MSM even though we prove that both methods have the same pruning power under L_2-norm.

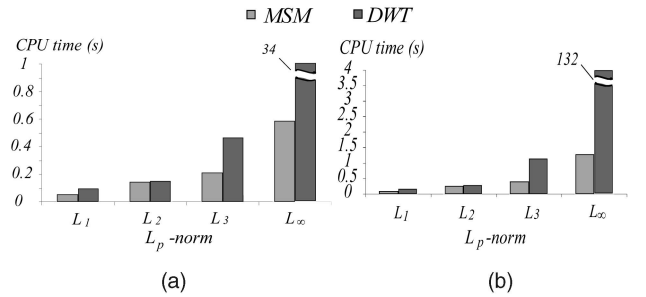


Fig. 7. Comparison between MSM and DWT (randomwalk). (a) Pattern length = 512. (b) Pattern length = 1,024.

Results from L_1-norm show that wavelet is not a good method to do filtering under L_1-norm. Since L_1-norm is not preserved by the wavelet transform, an L_2-norm distance has to be computed in order to do the filtering correctly (without introducing false dismissals) [33]. That is the reason that MSM representation is an order of magnitude better than DWT under L_1-norm, as shown in Fig. 6a. For L_3- and L_∞-norm, as mentioned in [33], although the wavelet cannot preserve L_p-norm for $p \neq 2$, the problem can be fixed by issuing range queries under L_2-norm with a much larger radius. In particular, for L_3-norm, the radius of the pattern matching query is set to $\sqrt{3} \cdot \varepsilon$, where ε is the original radius; similarly, for L_∞-norm, the radius is $\sqrt{w} \cdot \varepsilon$, where w is the length of patterns. Note that, in Fig. 6d, the Z-axis is in logarithmic scale, indicating that DWT is very inefficient in L_∞-norm.

Fig. 7 demonstrates the same set of experiments on *randomwalk* data set, where pattern length is 512 in Fig. 7a and 1,024 in Fig. 7b. The CPU time of DWT is always greater than that of MSM. In summary, with 1,000 patterns and hundreds of stream data, the MSM representation in all four cases can quickly detect similar patterns over stream time series data for less than a second on average under different L_p-norms, compared to DWT.

8.3 Performance of Detecting Dynamic Patterns

Next, we consider the case where both query and pattern time series come from streams. In particular, we evaluate the efficiency of our MSM representation as well as the SS filtering in detecting dynamic patterns, comparing with DWT. Similar to previous experiments, we perform the test on both real and synthetic data sets, that is, *stock* and *randomwalk*. Specifically, we randomly pick up 1,000 pattern (super-)sequences of length greater than 512, since we need to simulate the scenario in which pattern series of length 512 are also extracted from a stream time series. Furthermore, we use the rest of data as the query time series data. Similarly, for the *randomwalk* data set, we obtain 1,000 series of length greater than 512 (1,024) from the data set as streaming patterns, and the rest as query time series. For query time series, similar to previous settings, we set the size of the sliding window to be 1.5 times the pattern length. That is, we fix the window size to 768 for *stock* and 768 (1,536) for *randomwalk*. We simulate the scenario that, at each timestamp, sliding windows of both query and pattern time series shift one position, and thus new query and patterns are formed. In this case, instead of performing the similarity match from scratch, we utilize the result of distance computations at the previous timestamp and

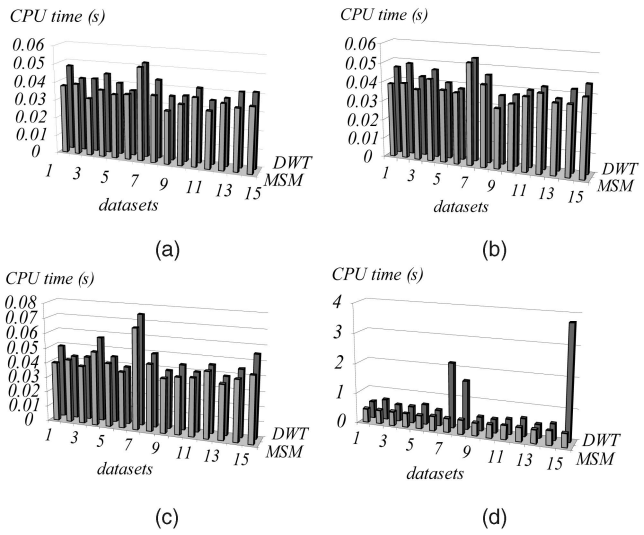


Fig. 8. Comparison between MSM and DWT (15 stock data sets, dynamic patterns). (a) L_1-norm. (b) L_2-norm. (c) L_3-norm. (d) L_∞-norm.

incrementally compute this distance. In particular, we store an array with real distances between two series computed at the previous timestamp. Then, at the current timestamp, we first check whether or not candidate pairs correspond to any entry in the array. If positive, we perform the incremental calculation; otherwise, the SS filtering is issued.

We compare the MSM representation with DWT using four distance functions L_1-, L_2-, L_3-, and L_∞-norms on stream pattern matching. In order to evaluate the efficiency, we use the same number of coefficients (level) in both MSM and DWT for the fair comparison. Fig. 8 illustrates the CPU time of MSM and DWT with 15 stock data sets under L_1-, L_2-, L_3-, and L_∞-norms. Among all these cases, MSM shows better performance than that of DWT. Since we consider 1,000 patterns and size of sliding windows 1.5 times the pattern size in both static and dynamic cases, the CPU time of results in Fig. 8 is comparable to that in Fig. 5. Obviously, the CPU cost in the dynamic scenario is order of magnitude lower than that with static patterns. The main reason is that, the incremental computation of one distance between series takes only $O(1)$ cost, which is much lower than that of either the SS filtering or computing the real distance from scratch. Therefore, our approach in the dynamic case is more efficient than that in the static one. Furthermore, we find that, with L_3-norm and L_∞-norm, the cost differences between the static and dynamic cases under DWT are much larger than that under MSM. This is reasonable in the sense that DWT always keeps more real distances at the last timestamp in the array than MSM does. Recall that, in order to perform queries with DWT under either L_3-norm or L_∞-norm, we have to enlarge the search range by increasing the radius up to either $\sqrt{3} \cdot \varepsilon$ or $\sqrt{w} \cdot \varepsilon$, respectively [33]. Since the number of candidates within the range increases, the number of real distances in the array also becomes greater, which leads to more incremental distance updates than MSM. Therefore, the reduced cost of DWT between dynamic and static case is more significant than that of MSM.

Fig. 9 illustrates the same set of experiments with dynamic patterns on the synthetic data set *randomwalk*,

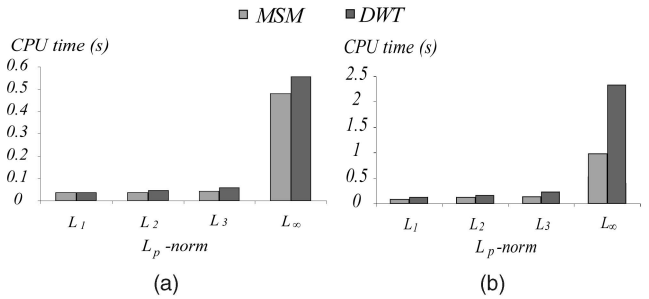


Fig. 9. Comparison between MSM and DWT (randomwalk, dynamic patterns). (a) Pattern length = 512. (b) Pattern length = 1,024.

where pattern length is 512 in Fig. 9a and 1,024 in Fig. 9b. The CPU time of DWT is always greater than that of MSM under various L_p-norms. Note that, although DWT under L_3- and L_∞-norms performs better than that in the static case, the space for the array is large.

9 Conclusions

In this paper, we have proposed a novel MSM representation together with a multistep filtering SS scheme, which facilitates detecting both static and dynamic patterns over time-series stream efficiently. The MSM representation can be incrementally computed with a low cost and enables us to prune out false alarms with no false dismissals. We show our analytic results on how to derive an optimal aborting level during multistep filtering. Compared to another popular multiscale representation, DWT, we prove that MSM has similar computational cost to DWT under L_2-norm. However, it is an order of magnitude better under other L_p-norms, since MSM is applicable to arbitrary L_p-norm, whereas it is not possible for DWT unless a very loose lower bound is used. We also discussed batch processing optimization and matching with dynamic patterns. Extensive experiments show that the multistep filtering SS on MSM offers an efficient and scalable methodology to detect patterns over stream time series data. In the future, it would be interesting to apply our MSM approximation as well as SS filtering techniques to answer pattern matching queries over predicted stream time series, which is very useful in applications like coal mine surveillance to report dangerous events even before the disaster occurs.

Acknowledgments

Funding for this work was provided by the Hong Kong RGC Grant 611907, National Grand Fundamental Research 973 Program of China under Grant 2006CB303000, and the NSFC Projects under Grants 60533110 and 60763001. Dr. Jinsong Han and Dr. Jian Ma were supported in part by Nokia APAC research grant.

References

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

[2] R. Agrawal, K.I. Lin, H.S. Sawhney, and K. Shim, "Fast Similarity Search in the Presence of Noise, Scaling, and Translation in Time-Series Database," *Proc. 21st Int'l Conf. Very Large Data Bases (VLDB)*, 1995.

[3] A.N. Akansu and R.A. Haddad, *Multiresolution Signal Decomposition.* Academic Press, 1992.

[4] D.J. Berndt and J. Clifford, "Finding Patterns in Time Series: A Dynamic Programming Approach," *Advances in Knowledge Discovery and Data Mining*, 1996.

[5] S. Berchtold, C. Bohm, and H.P. Kriegel, "The Pyramid-Technique: Towards Breaking the Curse of Dimensionality," *Proc. ACM SIGMOD*, 1998.

[6] J.S. Boreczky and L.A. Rowe, "Comparison of Video Shot Boundary Detection Techniques," *Proc. Eighth Int'l Symp. Storage and Retrieval for Image and Video Databases*, 1996.

[7] A. Bulut and A.K. Singh, "A Unified Framework for Monitoring Data Streams in Real Time," *Proc. 21st Int'l Conf. Data Eng. (ICDE)*, 2005.

[8] Y. Cai and R. Ng, "Indexing Spatio-Temporal Trajectories with Chebyshev Polynomials," *Proc. ACM SIGMOD*, 2004.

[9] K.P. Chan and A.W.-C. Fu, "Efficient Time Series Matching by Wavelets," *Proc. 15th Int'l Conf. Data Eng. (ICDE)*, 1999.

[10] L. Chen and R. Ng, "On the Marriage of Edit Distance and L_p Norms," *Proc. 30th Int'l Conf. Very Large Data Bases (VLDB)*, 2004.

[11] Q. Chen, L. Chen, X. Lian, Y. Liu, and J.X. Yu, "Indexable PLA for Efficient Similarity Search," *Proc. 33rd Int'l Conf. Very Large Data Bases (VLDB)*, 2007.

[12] L. Chen, M.T. Ozsu, and V. Oria, "Robust and Fast Similarity Search for Moving Object Trajectories," *Proc. ACM SIGMOD*, 2005.

[13] C. Cranor, T. Johnson, and O. Spatscheck, "Gigascope: A Stream Database for Network Applications," *Proc. ACM SIGMOD*, 2003.

[14] M. Dubinko, R. Kumar, J. Magnani, J. Novak, P. Raghavan, and A. Tomkins, "Visualizing Tags over Time," *Proc. Int'l Conf. World Wide Web (WWW)*, 2006.

[15] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos, "Fast Subsequence Matching in Time-Series Databases," *Proc. ACM SIGMOD*, 1994.

[16] L. Gao and X.S. Wang, "Continually Evaluating Similarity-Based Pattern Queries on a Streaming Time Series," *Proc. ACM SIGMOD*, 2002.

[17] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," *Proc. ACM SIGMOD*, 1984.

[18] A. Kemper and B. Stegmaier, "Evaluating Bestmatchjoins on Streaming Data," Technical Report MIP-0204, Universitt Passau, 2002.

[19] E. Keogh, "Exact Indexing of Dynamic Time Warping," *Proc. 28th Int'l Conf. Very Large Data Bases (VLDB)*, 2002.

[20] F. Korn, H. Jagadish, and C. Faloutsos, "Efficiently Supporting Ad Hoc Queries in Large Datasets of Time Sequences," *Proc. ACM SIGMOD*, 1997.

[21] M. Li and Y. Liu, "Underground Coal Mine Monitoring with Wireless Sensor Networks," *ACM Trans. Sensor Networks*, 2009.

[22] X. Lian, L. Chen, J.X. Yu, G. Wang, and G. Yu, "Similarity Match over High Speed Time-Series Streams," *Proc. 23rd Int'l Conf. Data Eng. (ICDE)*, 2007.

[23] J. Lin, E. Keogh, S. Lonardi, and B. Chiu, "A Symbolic Representation of Time Series, with Implications for Streaming Algorithms," *Proc. ACM/SIGMOD Int'l Workshop Research Issues on Data Mining and Knowledge Discovery (DMKD)*, 2003.

[24] Y. Liu, L. Xiao, X. Liu, L.M. Ni, and X. Zhang, "Location Awareness in Unstructured Peer-to-Peer Systems," *IEEE Trans. Parallel and Distributed Systems*, vol. 16, no. 2, pp. 163-174, Feb. 2005.

[25] Z. Liu, X. Yu, X. Lin, H. Lu, and W. Wang, "Locating Motifs in Time-Series Data," *Proc. Pacific-Asia Conf. Knowledge Discovery and Data Mining (PAKDD)*, 2005.

[26] A. Natsev, R. Rastogi, and K. Shim, "Walrus: A Similarity Retrieval Algorithm for Image Databases," *Proc. ACM SIGMOD*, 1999.

[27] S. Papadimitriou, J. Sun, and C. Faloutsos, "Streaming Pattern Discovery in Multiple Time-Series," *Proc. 31st Int'l Conf. Very Large Data Bases (VLDB)*, 2005.

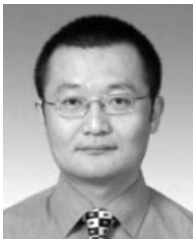
[28] E.J. Stollnitz, T.D. Derose, and D.H. Salesin, *Wavelets for Computer Graphics: Theory and Applications.* Morgan Kaufmann, 1996.

[29] M. Vlachos, G. Kollios, and D. Gunopulos, "Discovering Similar Multidimensional Trajectories," *Proc. 18th Int'l Conf. Data Eng. (ICDE)*, 2002.

[30] R. Weber, H.-J. Schek, and S. Blott, "A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces," *Proc. 24th Int'l Conf. Very Large Data Bases (VLDB)*, 1998.

[31] H. Wu, B. Salzberg, and D. Zhang, "Online Event Driven Subsequence Matching over Financial Data Streams," *Proc. ACM SIGMOD,* 2004.
[32] W. Xue, Q. Luo, L. Chen, and Y. Liu, "Contour Map Matching for Event Detection in Sensor Networks," *Proc. ACM SIGMOD,* 2006.
[33] B.-K. Yi and C. Faloutsos, "Fast Time Sequence Indexing for Arbitrary L_p-Norms," *Proc. 26th Int'l Conf. Very Large Data Bases (VLDB),* 2000.
[34] B.-K. Yi, H. Jagadish, and C. Faloutsos, "Efficient Retrieval of Similar Time Sequences under Time Warping," *Proc. 14th Int'l Conf. Data Eng. (ICDE),* 1998.
[35] B.-K. Yi, N. Sidiropoulos, T. Johnson, H.V. Jagadish, C. Faloutsos, and A. Biliris, "Online Data Mining for Co-Evolving Time Sequences," *Proc. 16th Int'l Conf. Data Eng. (ICDE),* 2000.
[36] Y. Zhu and D. Shasha, "Efficient Elastic Burst Detection in Data Streams," *Proc. Ninth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD),* 2003.
[37] Y. Zhu and D. Shasha, "Warping Indexes with Envelope Transforms for Query by Humming," *Proc. ACM SIGMOD,* 2003.
[38] V. Megalooikonomo, Q. Wang, G. Li, and C. Faloutsos, "A Multiresolution Symbolic Representation of Time Series," *Proc. 21st Int'l Conf. Data Eng. (ICDE),* 2005.

Xiang Lian received the BS degree from Nanjing University in 2003. He is currently a PhD candidate in the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Kowloon, Hong Kong. His research interests include stream time series and uncertain databases. He is a student member of the IEEE.



Lei Chen received the BS degree in computer science and engineering from Tianjin University, China, in 1994, the MA degree from the Asian Institute of Technology, Thailand, in 1997, and the PhD degree in computer science from the University of Waterloo, Canada, in 2005. He is currently an assistant professor in the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Kowloon, Hong Kong. His research interests include uncertain databases, graph databases, multimedia and time series databases, and sensor and peer-to-peer databases. He is a member of the IEEE.

Jeffrey Xu Yu received the BE, ME, and PhD degrees in computer science from the University of Tsukuba, Japan, in 1985, 1987, and 1990, respectively. He was a research fellow (April 1990-March 1991) and an assistant professor (April 1991-July 1992) in the Institute of Information Sciences and Electronics, University of Tsukuba, a lecturer in the Department of Computer Science, Australian National University (July 1992-June 2000). He is currently a professor in the Department of System Engineering, The Chinese University of Hong Kong, Shatin, NT, Hong Kong. His major research interests include data mining, data streaming mining and processing, XML query processing and optimization, and graph database. He is a senior member of the IEEE and a member of the IEEE Computer Society.

Jinsong Han received the BS degree in computer science from Shandong University of Technology, China, in 1997, the MEng degree in computer science from Shandong University, China, in 2000, and the PhD degree in computer science and engineering from Hong Kong University of Science and Technology (HKUST), Kowloon, Hong Kong, in 2007. He is currently a postdoctoral fellow in the Department of Computer Science and Engineering, HKUST. His research interests include peer-to-peer computing, anonymity, network security, pervasive computing, and high-speed networking. He is a member of the IEEE, the IEEE Computer Society, and the ACM.

Jian Ma received the BSc and MSc degrees from Beijing University of Posts and Telecommunications in 1982 and 1987, respectively, and the PhD degree from Helsinki University of Technology in 1994. He is currently a principal member of the research staff at the Nokia Research Center, Beijing, where he is responsible for university cooperation in China and other APACs, Nokia Postdoc Working Station, and Nokia-Tsinghua Joint Research Lab. He has 11 patents granted and tens of patent applications. He has authored or coauthored more than 150 publications in journals and conferences, and a couple of books and book chapters. He is also an adjunct professor at Beijing Posts and Telecommunications, Institute of Computing Technology in Chinese Academy of Science, Graduation University of Chinese Academy of Science, Tongji University, and Beihang University. His current research interests include consumer services on wireless sensor networks, mobile Internet applications, pervasive computing, and ad hoc and P2P networking.