2013

PRESEE: An MDL/MML Algorithm to Time-Series Stream Segmenting

Kaikuo Xu
Chengdu University of Technology

Yexi Jiang
School of Computing and Information Sciences, Florida International University, yjian004@fiu.edu

Mingjie Tang
Purdue University

Changan Yuan
Guangxi Teacher's College

Changjie Tang
Sichuan University

Recommended Citation
Kaikuo Xu, Yexi Jiang, Mingjie Tang, Changan Yuan, and Changjie Tang, "PRESEE: An MDL/MML Algorithm to Time-Series Stream Segmenting," The Scientific World Journal, vol. 2013, Article ID 386180, 11 pages, 2013. doi:10.1155/2013/386180

Hindawi Publishing Corporation
The Scientific World Journal
Volume 2013, Article ID 386180, 11 pages
http://dx.doi.org/10.1155/2013/386180

Research Article

PRESEE: An MDL/MML Algorithm to Time-Series Stream Segmenting

Kaikuo Xu,[1] Yexi Jiang,[2] Mingjie Tang,[3] Changan Yuan,[4] and Changjie Tang[5]

[1] *College of Computer Science & Technology, Chengdu University of Information Technology, Chengdu 610225, China*
[2] *School of Computing and Information Sciences, Florida International University, Miami, IN 33199, USA*
[3] *Department of Computer Science, Purdue University, West Lafayette, FL 47996, USA*
[4] *Guangxi Teachers Education University, Nanning 530001, China*
[5] *School of Computer Science, Sichuan University, Chengdu 610065, China*

Correspondence should be addressed to Changan Yuan; yca@gxtc.edu.cn

Received 31 March 2013; Accepted 9 May 2013

Academic Editors: R. Haber, S.-S. Liaw, J. Ma, and R. Valencia-Garcia

Time-series stream is one of the most common data types in data mining field. It is prevalent in fields such as stock market, ecology, and medical care. Segmentation is a key step to accelerate the processing speed of time-series stream mining. Previous algorithms for segmenting mainly focused on the issue of ameliorating precision instead of paying much attention to the efficiency. Moreover, the performance of these algorithms depends heavily on parameters, which are hard for the users to set. In this paper, we propose *PRESEE* (*parameter-free, real-time, and scalable time-series stream segmenting algorithm*), which greatly improves the efficiency of time-series stream segmenting. PRESEE is based on both MDL (minimum description length) and MML (minimum message length) methods, which could segment the data automatically. To evaluate the performance of PRESEE, we conduct several experiments on time-series streams of different types and compare it with the state-of-art algorithm. The empirical results show that PRESEE is very efficient for real-time stream datasets by improving segmenting speed nearly ten times. The novelty of this algorithm is further demonstrated by the application of PRESEE in segmenting real-time stream datasets from ChinaFLUX sensor networks data stream.

1. Introduction

Time series stream is everywhere in our daily life. It is widely used in fields such as ecology, medical care, and environment. These applications make time series stream type be possibly the most frequently encountered type for data mining problems [1]. Hence, in recent years, a large number of works focus on time series stream mining.

In order to process massive data efficiently, the method of time series stream segmenting is employed. The primary purpose of time series segmenting is dimensionality reduction. To achieve the goal of accelerating later mining tasks, time-series stream segmenting decomposes the time series stream into smaller number of segments. After segmenting, each segment can be described by a simple model like linear segment and monotonic segment [2]. An example of time-series segmenting can be seen in Figure 1.

There are several time series stream fitting models proposed, including symbolic mappings [3], adaptive multivariate spline [4], hybrid adaptive [5], wavelets [6], Fourier transforms [7], and piecewise linear representation [8, 9]. However, neither of them could handle different types of time series streams or is parameter free.

For real-time series application, the algorithm should be able to handle continuously real-time stream, which means that the stream could only be scanned once. A lot of real applications such as sensor network data [10], stock market trading data [11], or intensive-care unit (ICU) data [12] are in this form since the data are generated very fast and the processing time is limited. So, for time series stream segmentation, the issues of scalability, numerical stability, and efficiency cannot be avoided.

In this paper, we propose PRESEE to segment time series stream based on MDL/MML method [13, 14]. MDL/MML is

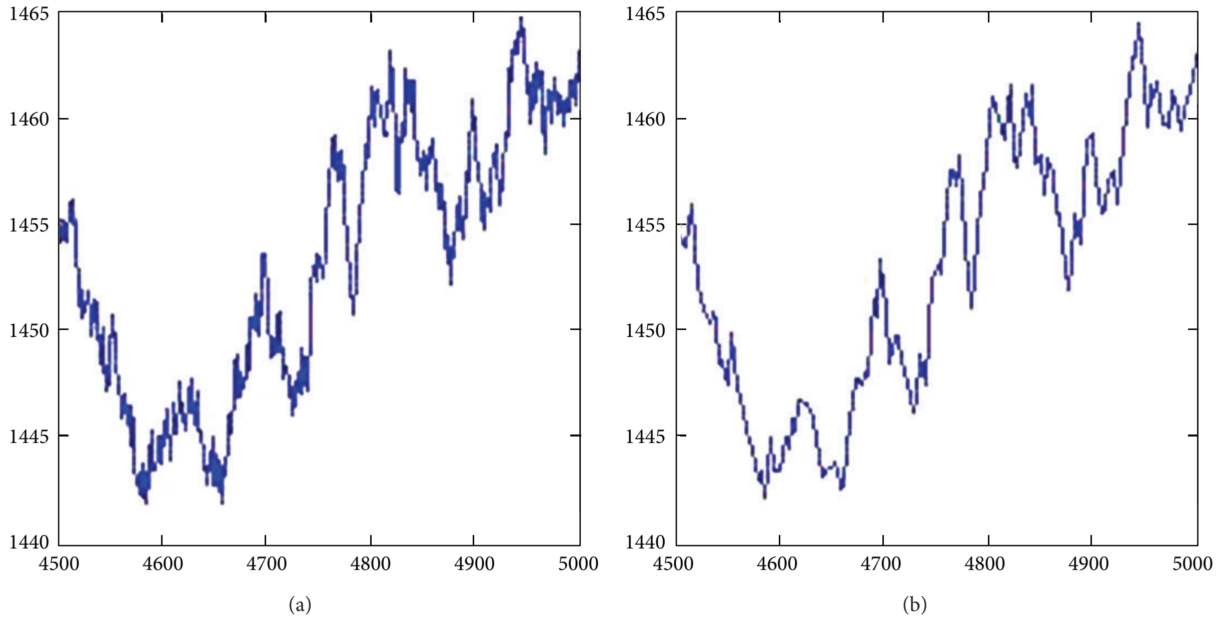


Figure 1: Time series original data (a) and its segmenting result (b).

an information expressing method in the field of information theory. By capturing the characteristics of information distribution in data, it can reduce the size of data while retaining most of the critical information. PRESEE the following characteristics has

(1) High scalability. It can process time series stream in linear time. PRESEE adopts slide window to process data with the size of gigabytes or even larger scale.

(2) Parameter free. Parameter settings are not essential in PRESEE for an entry-level user. This may avoid the trouble of misleading the algorithm by setting any improper parameters. Of course, if the end users are the domain experts and have confidence to set proper parameters, they can set some optional parameters to accelerate the segmenting speed.

(3) Adaptive. PRESEE can segment the time series data according to the characteristic of data. Since the segmenting strategy is based on MDL/MML, it can segment time series automatically. Violating place requires more characteristic points while elsewhere requires less.

(4) Pipeline. PRESEE can output the earlier data while processing the newly arrived data. Thus, the later time series stream mining algorithm and PRESEE can run simultaneously.

The rest of this paper is organized as follows. The related work is described in Section 2. Some necessary concepts are introduced in Section 3. A time series stream segmenting algorithm named PRESEE is presented in Section 4. The result of experiments is evaluated in Section 5. Finally, the paper is concluded and future work is discussed in Section 6.

2. Related Work

2.1. Time Series Stream Mining. Time series stream mining is possibly the most frequent mining task in recent data mining community. In particularly, in the last several years, a large number of papers are related to this area [15–17]. Time series stream mining derives from traditional time-series mining [1–4, 11, 18]. As a further requirement of deep understanding of the time series, it turns into high-dimensional data mining problem.

2.1.1. Segmenting. Segmenting is one of the major tasks in time-series stream mining. In order to process time-series data efficiently and effectively, segmenting is a key step for other time-series mining tasks. A lot of algorithms focus on finding good global segmenting of the time-series data.

There are mainly three characteristics of these algorithms. Firstly, these methods are mainly based on dynamic programming [19, 20], top-down [21], and bottom-up [22] strategies. Secondly, they require domain expert knowledge to set the parameters, either the parameter to measure the error [2, 22] or parameter k ($k \ll n$) to control the number of segments [19, 21]. Thirdly, these algorithms can at most handle millions of data, and they can hardly to handle stream data (gigabytes at least) due to the limitation of the algorithms.

Segmenting with slide window can handle large-scale data. This method is attractive because it can be easily implemented as an online algorithm. Some existing slide-window-based algorithms work well, but their performances are parameter dependent. Since different time-series data types such as electrocardiogram (ECG), water level, and stock market own quite different characteristics, it is hard to find a general set of parameters for all these data types.

2.1.2. MDL/MML. The theory of minimum message length (MML) and minimum description length (MDL) first appears in the computation complexity community [23, 24] then in the categorization community [25]. Its application in data mining community is the work of climate data segmentation [26], trajectory clustering [27], and social network mining [28]. So far, to the best of our knowledge, our work segmenting time-series stream with MDL/MML is the work with the most features.

3. Preliminary

This section reviews the concepts for time-series data mining. Section 3.1 introduces terminology about the time series. Section 3.2 presents the distance function used in this paper. Section 3.3 is the problem statement.

3.1. Terminology. We first begin with the definition of the time-series data type.

Definition 1. Time-series: let R_d denote a set of the observed values for given variables in the research domain. Let $s_i \in R_d$ be an element observed at time i. Time-series $S = \langle s_1 s_2 s_3 \cdots s_n \rangle$ is an ordered sequence of n such elements. From the stream view, the length of S is infinite.

Slide window may be a general and effective way to handle massive data that cannot be processed in whole. Thus we employ slide window idea to do the segmenting task.

Definition 2. Slide window: let B be a user-defined buffer to hold elements and w be the size of elements that B can hold. The slide window $W = \langle w_1 w_2 \cdots w_w \rangle$ is the buffer to hold a continuous subsequence of S at any time. All the data in slide window can be processed by the algorithm in one time.

3.2. Distance Function for Time Series Segments. For the ease of segmenting, some data transformation work should be done. Almost all kinds of time series data can be discretized and transformed in the form of lines. For example, the original time-series data $S = \langle s_1 s_2 \cdots s_n \rangle$ can be discretized into $n-1$ lines: $s_1 s_2, s_2 s_3, \ldots, s_{n-1} s_n$. The goal of segmenting is to generate m ($m < n$) lines that can represent most of the characteristics of original lines. There should be a distance function to measure the distance between the original time-series line L_o and the candidate segment L_s. In order to better measure the distance between original time-series stream and its segmenting result, firstly, the distance function should be simple so that the stream can be processed very fast. Additionally, the measurement should consider the shape of stream and its segmenting result. Finally, the focus of factor in measurement can vary according to different application. After delving into the character of time-series data, we find that the best way to measure the distance between time-series by considering the conciseness and preciseness is to use *Hausdorff metric*. *Hausdorff metric* has been previously used in the area of pattern recognition and trajectory mining [27, 29]. Previous works proved that it is precise in the scenario of shape similarity measurement. In the scenario of time-series segmenting, we represent the *segmenting distance* by considering the perpendicular and angle space relationship based on *Hausdorff metric*.

Segmenting distance is a quantitative criterion to measure the quality of segmenting. Smaller distance represents better segment result for the original stream. The final form of distance between the original line and the segment it belongs to is defined in Definition 3.

Definition 3. Segmenting distance. Let L_o be the original line, L_s be the candidate segment, l_{p1} and l_{p2} be the distances from the start point and end point of L_o to L_s, respectively (Formula (1)), and $\theta(L_o, L_s)$ be the smaller intersection angle between two lines. Then the following can be considered.

(1) The perpendicular distance between two lines is defined as $d_p(L_o, L_s)$ in Formula (2). In Formula (1), (x_{ps}, y_{ps}) and (x_{pe}, y_{pe}) represent the start point and end point of each original time-series line, respectively; (x_s, y_s) and (x_e, y_e) represent the coordinates of the start point and end point of a candidate segment time-series line (one possible segment solution in the process of segmenting computation), respectively.

(2) The angle distance between two lines is defined as $d_a(L_o, L_s)$ in Formula (3).

(3) The segmenting distance between two lines is defined as $d(L_o, L_s)$ in Formula (4): the weighted sum of perpendicular distance and angle distance.

Consider

$$l_{p1(2)} = \frac{\left| k \times x_{ps(pe)} - y_{ps(pe)} + b \right|}{\sqrt{k^2 + 1}}, \quad \left(k = \frac{y_s - y_e}{x_s - x_e}, b = y_s - k \times x_s \right), \tag{1}$$

$$d_p(L_o, L_s) = \frac{l_{p1}^2 + l_{p2}^2}{l_{p1} + l_{p2}}, \tag{2}$$

$$d_a(L_o, L_s) = L_o \times \sin(\theta(L_o, L_s)), \tag{3}$$

$$d(L_o, L_s) = w_p \cdot d_p + w_a \cdot d_a. \tag{4}$$

The sum of weight w_p and w_a should be 1, and they can both be set to 1/2 if there is no special requirement. Figure 2 and Example 4 show an example of how to compute the distance.

Example 4. As shown in Figure 2, there are 3 original lines L_{o1}(line $s_1 s_2$), L_{o2}(line $s_2 s_3$), and L_{o3}(line $s_3 s_4$) and one segment L_s(line $s_1 s_4$). Since it can be observed that the start point of L_{o1} and L_s is the same point, the distance $d(L_{o1}, L_s) = w_p \cdot l_{p2} + w_a \cdot (L_o \cdot \sin\theta_1)$. The distance between L_{o2} and L_s is $d(L_{o2}, L_s) = w_p \cdot (l_{p2}^2 + l_{p3}^2)/(l_{p2} + l_{p3}) + w_a \cdot (L_o \cdot \sin\theta_2)$, and between L_{o3} and L_s is $d(L_{o3}, L_s) = w_p \cdot l_{p3} + w_a \cdot (L_o \cdot \sin\theta_3)$.

3.3. Problem Statement. Given a time series S with length n (i.e., $S = \langle s_1 s_2 s_3 \cdots s_n \rangle$, n can be infinite), our algorithm

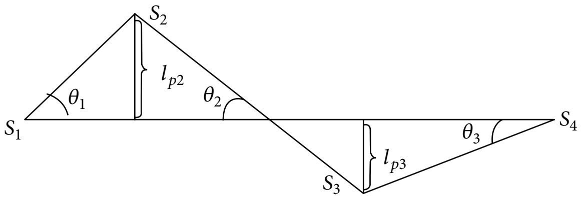


FIGURE 2: Distance between 3 original lines and one segment that they belong to.

generates a sequence of character points $C = \langle c_1, c_2, c_3 \dots c_m \rangle$. For sequence C, each pair c_i to s_j has a projection relationship: $g(i) = j$. This means that each c_i located at i in C has a counterpart located at j in S. For each consecutive character point $c_x, c_y \in C$, there exist several points $(s_i, \dots, s_j)$ in S such that $g(x) < i < j < g(y)$. Every pair of c_x, c_y represents a segment which is an approximation of lines represented by several pairs of consecutive points in the original time series stream S. Thus, these m character points partition the original stream into $m-1$ continuous segments. And for each $s_x \in S$, if s_x is just the start or end point of one segment, it belongs to two segments; otherwise, it only belongs to one segment. Figure 3 shows lines representing S compared with lines representing C.

The segmenting algorithm is implemented under a pipeline framework shown in Figure 4. Besides the segmenting algorithm, we had already implemented the time-series stream motif mining algorithm. This framework is designed specifically for handling time-series stream mining. It owns several advantages as follows.

(1) Data stream is only scanned once. When data flow out of the slide window, it would never turn back to slide window again.

(2) Mining tasks can be processed simultaneously. Earlier data that have been segmented before can be processed by following mining task while the later data is under processing by segmenting task.

4. PRESEE Algorithm

This section first introduces how to use segmenting strategy based on MDL/MML in our algorithm PRESEE, and then introduces this algorithm in detail.

4.1. Information-Theory-Based Segmenting Strategy. Our algorithm aims at finding the best segments for time-series. As for the problem of segmenting, there are two properties to measure the quality: *preciseness* and *conciseness*. *Preciseness* measures the distance between the lines represented by consecutive points of the character point c_i, c_j in set C and lines represented by consecutive points of original time series in stream S between the corresponding two character points c_i, c_j. Smaller distance indicates better *preciseness*. *Conciseness* measures how less the character points are used todepict the certain length of data points in original stream. Less character points represents better conciseness.

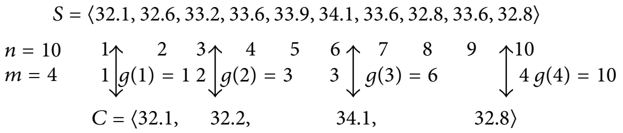

$S = \langle 32.1, 32.6, 33.2, 33.6, 33.9, 34.1, 33.6, 32.8, 33.6, 32.8 \rangle$
$n = 10$
$m = 4$
1 2 3 4 5 6 7 8 9 10
1 $g(1) = 1$
2 $g(2) = 3$
3 $g(3) = 6$
4 $g(4) = 10$
$C = \langle 32.1,$ $32.2,$ $34.1,$ $32.8 \rangle$

FIGURE 3: The comparison between original time-series S and the character points C.

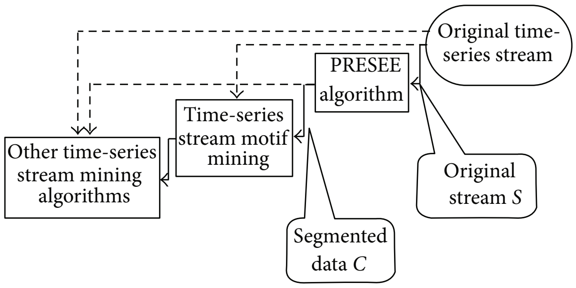

Original time-series stream
PRESEE algorithm
Time-series stream motif mining
Other time-series stream mining algorithms
Original stream S
Segmented data C

FIGURE 4: The pipeline for time-series mining. All data are processed as stream. At first the original data are flowed into PRESEE and then flowed into any other time-series stream mining algorithms.

It is easy to get the conclusion that, when every point in original stream is the character point, *preciseness* gets its maximum. However, such kind of segmenting is meaningless since it in fact just lets the stream go through the slide window and does not do any work to compress the stream. *Conciseness* reaches the maximum when there are only two character points for the stream, the start point and the end point.

The best *preciseness* and best *conciseness* cannot be satisfied at the same time because they are contradictory. Therefore, we need to do some work to find the optimum tradeoff betwee*n preciseness* and *conciseness*, which generates the best segmenting L_{opt}.

In order to find the optimum tradeoff, we intend to solve this problem in information perspective by employing the MDL/MML principle in information theory area. We use MDL/MML in our algorithm because it is parameter free. MDL/MML can automatically find a proper estimate of original information. If no proper segmenting solution exists, the data S are deemed as random data. In our scenario, we simply keep all the information.

The code of MDL/MML is composed of two parts: $I = H : D$ [30, 31]. H specifies the hypothesis about the information (normally selected from a limited set of possible hypotheses), while D specifies the code for the information based on the hypothesis. The shortest code I in all the H and D combinations are the optimum solution for the piece of information. In our scenario, it are the optimum for stream segmenting.

The cost of code is represented by its length. In *Shannon's theory*, the length of coding an event E in optimum condition is given by $-\log_2(E)$. In time-series segmenting scenario, the computation of the formula is as follows:

$$L(H) = \sum_{i=1}^{w-1} \log_2 \left(len \left(s_{c_i} s_{c_{i+1}} \right) \right), \quad (5)$$

$$L(S \mid H) = \sum_{i=1}^{w-1} \sum_{k=c_i}^{c_{i+1}-1} \left\{ \log_2 w_p d_p \left(s_{c_i} s_{c_{i+1}}, s_k s_{k+1} \right) + \log_2 w_a d_a \left(s_{c_i} s_{c_{i+1}}, s_k s_{k+1} \right) \right\}, \tag{6}$$

$$L_{\text{opt}} = \operatorname{argmin} \{ L(H) + L(S \mid H) \}. \tag{7}$$

In the first two formulas, w represents the size of data S, $c_x \in C$ represents the character points. The optimum segmenting is the minimum value of sum of $L(H)$ and $L(S \mid H)$. The following is a concrete computation example for Figure 3. Line $s_1 s_4$ is the optimum segmenting for point s_1 through s_4.

Consider

$$L(H) = \log_2 \left(len \left(s_1 s_4 \right) \right),$$

$$\begin{aligned} L(S \mid H) &= \left\{ \log_2 \frac{1}{2} d_p \left(s_1 s_2, s_1 s_4 \right) + \log_2 \frac{1}{2} d_a \left(s_1 s_2, s_1 s_4 \right) \right\} \\ &\quad + \left\{ \log_2 \frac{1}{2} d_a \left(s_2 s_3, s_1 s_4 \right) + \log_2 \frac{1}{2} d_p \left(s_2 s_3, s_1 s_4 \right) \right\} \\ &\quad + \left\{ \log_2 \frac{1}{2} d_a \left(s_3 s_4, s_1 s_4 \right) + \log_2 \frac{1}{2} d_p \left(s_3 s_4, s_1 s_4 \right) \right\}. \end{aligned} \tag{8}$$

4.2. Algorithm Details. Finding global optima requires computing all partitions possibilities of the points, which is prohibitive for real applications. We present a greedy algorithm to find local optima.

Algorithm 1 shows the details of the segmenting process. At first, only the data flowed into slide window are processed. In lines 5 and 6, the costs of MDL_{seg} and $\text{MDL}_{\text{noseq}}$ are computed, respectively. $\text{MDL}_{\text{seq}}(s_i, s_j)$ denotes the MDL cost by considering s_i and s_j as the character points for points s_x $(i < x < j)$. It equals to $L(H) + L(D \mid H)$. $\text{MDL}_{\text{noseq}}(s_i, s_j)$ denotes the MDL cost by assuming that there are no character points between s_i and s_j. It equals to $L(H)$. In the greedy strategy, the local optimum solution is the longest segment that satisfies the inequality (9).

In the algorithm shown previously, the points in slide window are scanned sequentially only once. The candidate segmenting (segment with $s_{startIndex}$ and $s_{curIndex}$ as start point, and end point resp.) grows once per time to test whether it satisfies inequality (9).

There is a parameter *batchSize* for this algorithm. The default value is 1, and the user can set it as a larger integer. The algorithm will return *batchSize* + 1 character points per time. Thus, the algorithm can run faster.

Consider

$$\text{MDL}_{\text{seq}} \left(s_i, s_j \right) < \text{MDL}_{\text{noseq}} \left(s_i, s_j \right). \tag{9}$$

PRESEE algorithm calls Algorithm 1 every time when slide window is full. Algorithm 2 describes PRESEE algorithm in the form of pseudocode.

In line 2, the slide window is filled at the first time. Then the method *ReadIn*() is called in *while* loop. *ReadIn*() takes the response of filling slide window and checking whether there is new data. It returns *false* when no new data exists. In line 4, Algorithm 1 is called to provide the local optima segmenting result based on the data in slide window. It is possible that no proper segment exists. Thus the size of *tmpSet* is less than 2. In this scenario, we simply put all the data in slide window into *apprSet* and empty the slide window. Otherwise, add *at most* first *batchSize* + 1 points in *tmpSet* into *apprSet*. We use “at most” here because it is possible that all the points in slide window may be generated less than in *batchSize* segments.

From the pseudocode, we can see that the data are input and output simultaneously (line 4 and line 13), which guarantees that the earlier data can be processed by later mining algorithm. Additionally, it is obvious that the stream is only scanned once and processed once. Thus the time complexity of both algorithms is $O(n)$, where n is the length of time-series stream.

5. Empirical Comparison of the Segmenting Algorithms

In this section, we demonstrate the effectiveness and efficiency of the proposed algorithm through several sets of experiments on large collections of real and synthetic time-series datasets. For the effectiveness test, the precision of the proposed method is compared with nonstream segmenting algorithm. Then the speed and scalability of the algorithm are tested with a different scale of datasets ranging from 10 M to 10 G.

All the experiments are performed on a laptop computer with 2 GHz Intel Core 2 Duo CPU and 3G main memories. The C++ implementation of the algorithm and the related source code are all available at http://code.google.com/p/ots-sm/.

5.1. Benchmark Algorithm. The performance of the proposed algorithm is compared with well-known benchmark algorithms. A good candidate is the BU (bottom-up) algorithm [22]. It is used as a counterpart in precision algorithm. Since BU cannot handle stream-like datasets and the time complexity is uncertain for large datasets, we use slide-window-based bottom-up (SWBU for short) segmenting algorithm in the part of efficiency and scalability experiment instead.

5.2. Dataset. Real Datasets. We consider two sets of real datasets to evaluate our algorithm. The first set includes three classical datasets: IBM stock price dataset from 1.2.1961 to 1.6.2010 [32], “Dodgers,” and “ICU” from UCI Machine Learning Repository [33] used by most of the papers in data mining and machine learning community. The second set of real-time stream data is collected from Chinese Terrestrial Ecosystem Flux Research Network (ChinaFLUX) [34], which is a long-term national network of micrometeorological flux measurement sites that measures the net exchange of carbon dioxide, water vapor, and energy. In this paper, we only choose part of the flux data from one wild filed survey site

```
Input: points in slideWindow Seg = $\langle p_1, p_2, p_3 \cdots p_{len}\rangle$,
       batchSize
Output: character points set $C = \langle c_{i1}, c_{i2} \cdots \rangle$
Put $p_1$ into C
curSize = 0, startIndex = 1, length = 1;
While startIndex + length < len do
    curIndex = startIndex + length;
    $Cost_{\text{seg}} = \text{MDL}_{\text{seg}}(S_{startIndex}, s_{curIndex})$; $Cost_{\text{noseg}} =$
    $\text{MDL}_{\text{noseg}}(c_{startIndex}, c_{curIndex})$;
    If $cost_{\text{seg}} > cost_{\text{noseg}}$ then
      Put $p_{curIndxe-1}$ into C;
      ++curSize;
       If curSize == batchSize then
       // enough batch has been processed
          Return C;
       Else startIndex = curIndex − 1; length = 1;
     Else length = length + 1;
     If C has only one point do
       Return NULL;
     Else Put$_{plen}$ into C;
```

Algorithm 1: Segment in slide window.

```
Input: windowSize, $S = \langle s_1, s_2 \cdots \rangle$, batchSize
Output: $C = \langle c_1, c_2 \cdots \rangle$
slideWindow = {}, apprSet = {}
Read data into slideWindow
While ReadIn() do
    tmpSet = MDLSlideWindow(slideWindow,
    batchSize);
    If tmpSet.size() < 2 do
    // no proper hypothesis is found,
          data deemed as random noise
       Add all data in slideWindow into apprSet
       Empty slideWindow
  Else
       Add at most first batchSize + 1 points in
       tmpSet into apprSet
       Take out used points from slideWindow
       Output apprSet
```

Algorithm 2: PRESEE.

located in Yucheng, Shandong, China. The data is stored from 2009-03-04 to 2009-11-12 with the number of 38850980.

Synthetic Datasets. The synthetic datasets are generated according to Formula (10) with parameters in Table 1. The data is in the format of index, value, where index represents the timestamp of the record. The monotonically increasing record serial number is also available. In our experiments, without loss of generalization, we simply use the latter one.

For synthetic data, the range of values is bounded within [*lb*, *ub*]. The current record value fluctuates (increases, decreases, or remains the same) according to previous record value.

Consider

$$\Delta = \text{sign} \times (\text{random}() \bmod (lb - \text{previous\_value})) \times \text{sharpness}. \tag{10}$$

The *sign* is randomly selected as either + or −, and *sharpness* is a parameter to control the power of fluctuation. It is easy to observe that when the value would suffer more resistance when it goes far away from the mean $((ub - lb)/2)$.

Table 1: Parameters for synthetic datasets.

Number	Data size	Lower bound (lb)	Upper bound (ub)	Sharpness
1	10,000 (10 k)	0	3000	0.001
2	100,000 (100 k)	0	3000	0.001
3	1,000,000 (1 M)	0	3000	0.001
4	10,000,000 (10 M)	0	3000	0.001
5	10,000,000 (10 M)	0	3000	0.002
6	10,000,000 (10 M)	0	3000	0.0005
7	10,000,000 (10 M)	0	3000	0.0002
8	10,000,000 (10 M)	0	3000	0.0001
9	100,000,000 (100 M)	0	3000	0.00005
10	1,000,000,000 (1 G)	0	5000	0.00001
11	10,000,000,000 (10 G)	0	5000	0.00001

6. Results

Visualization of Segmenting Result. For the ease of observing the segmenting result, Figure 5 presents the visualization of three real datasets (IBM stock, ICU, and Dodger). The first row is the original datasets; the second row is the segmenting result generated by BU; and the last row is generated by PRESEE. It is obvious that the charts in the first row seem to be the most complex because they contain the most detailed information about the time-series. For the other two rows, the charts seem to be more concise because segmenting algorithm removes some of the unimportant information from original time-series. Nevertheless, the main trends in the charts are reserved. This means that both algorithms keep the characteristics of original time-series data.

Precision. We compare our algorithm with BU, which can process general types of time-series datasets and can find the global optimum segmenting solution. We evaluate the result of segmenting algorithm via the *error rate* measure in Formula (11). Let w be the number of segments and n be the number of points in segment i. The error rate is computed as follows:

$$\text{error\_rate} = \frac{1}{w}\sum_{i=1}^{i<w}\frac{1}{n}\sum_{o_j \in s_i} d\left(L_{o_j}, L_{s_i}\right). \quad (11)$$

Error rate for fifteen datasets is reported in Table 2. It is evident that, for generating approximately the same number of segments, the error rate of bottom-up and PRESEE stays in the same level. That means that PRESEE can generate segments based on partial data (data only in slide window) no worse than segments generated in global perspective.

Problem of Getting Proper Number of Segments. In Table 2, for the real datasets ChinaFLUX, BU fails to find a proper number of segments because this kind of data changes tremendously. This is the flaw of BU. One significant advantage of PRESEE over BU is that users do not need to set any threshold parameter. The parameter of BU is hard to set. A little deviation would generate a quite different number of segments. In order to find out the relationship between segment size and the parameter error threshold that BU requires, we run each dataset 100 times to find a proper parameter that can generate the same number of segments as PRESEE. Figure 6 shows their relationship for BU. In the experiment, all three real datasets encounter a big drop of segments number when the error threshold increases. In particular, for Dodgers datasets, the number of segments drops from 6082 to 1958 when the value changes from 1.0 to 1.1, which is very significant. Further experiments show that, even we only increase the threshold with 0.001, the change is also tremendous. Such phenomenon indicates that we should be careful to the error threshold parameter of BU. Such puzzle can be well avoided by the user of PRESEE.

Efficiency and Scalability. We only test the relative speed of algorithms since the absolute speed (running time) is varied according to machines. The speed of synthetic-dataset-generated algorithm is used as a benchmark. It reflects the maximum processing speed that a certain running machine can reach.

Since the IBM stock, ICU, and Dodger datasets are too small and not suitable for horizontal comparison in efficiency and scalability test, we use real-time ChinaFLUX datasets and the synthetic datasets 1–4 and 9–11 in this experiment. At first, we compare the efficiency among data generator, SWBU, and PRESEE. The error rate threshold of SWBU set as 1.1 means that SWBU can generate comparatively the same number of segments with PRESEE. Figure 7 shows the efficiency of different algorithms in logarithmic plot for synthetic datasets 1–4 and 9–11. It is certain that data generator owns the best speed since it just generates data without any extra processing. In this figure, we can find that the curve of PRESEE is very near to the curve of data generator and these two are far from the curve of SWBU (near one order of magnitude). Figure 8 shows its efficiency on different datasets (datasets ChinaFLUX). Figure 8 indicates that the efficiency would not be affected by characters of datasets. Table 3 shows the relative speed of two segmenting algorithms' relative speed (with t_{seg} representing the time cost of segmenting algorithm and t_{gen} representing the time cost of data generator). PRESEE is just a little slower than data generator with the value 1.1311, while SWBU is nearly one order of magnitude slower. This is because PRESEE is an $O(n)$

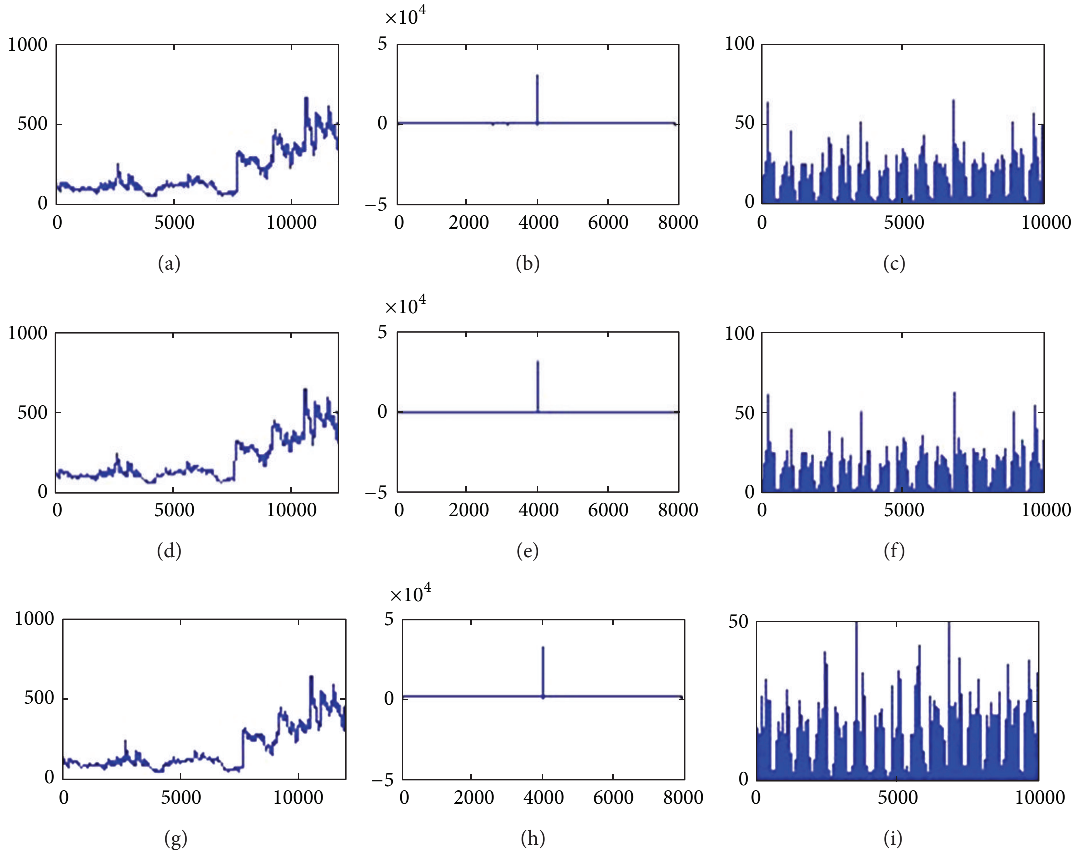


Figure 5: Time-series streams with their segmenting results: (a) IBM stock price, (b) ICU, and (c) Dodger. (d)–(f) Segmenting result with bottom-up. (g)–(i) Segmenting result with PRESEE.

Table 2: Precision of datasets.

Number	Data size	Error rate (BU)	Error rate (PRESEE)
1	10,000 (10 k)	2.00201	2.58561
2	100,000 (100 k)	1.75349	2.45916
3	1,000,000 (1 M)	1.67456	4.03172
4	10,000,000 (10 M)	1.65483	2.28419
5	10,000,000 (10 M)	1.66074	2.28695
6	10,000,000 (10 M)	1.65577	2.28768
7	10,000,000 (10 M)	1.65978	2.28862
8	10,000,000 (10 M)	1.66243	2.28124
9	100,000,000 (100 M)	1.67212	2.29100
10	100,000,000 (1 G)	N/A	2.27872
11	100,000,000 (10 G)	N/A	2.27984
12	IBM stock price from 1.2.1961 to 1.6.2010 (12087 records)	1.57696	2.38069
13	ICU (7931 records)	0.696307	0.893313
14	Dodger (10082 records)	1.34914	1.08395
15	ChinaFLUX (38850980 records)	N/A	1.76781

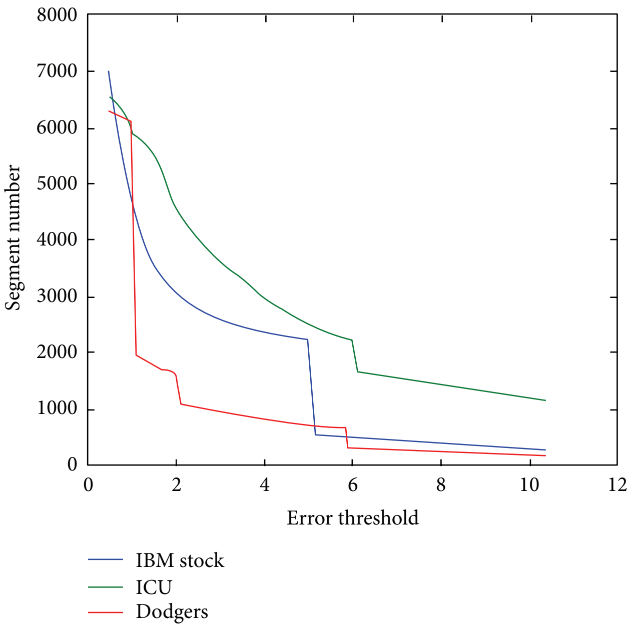


Figure 6: Relative of error threshold and segments number for BU.

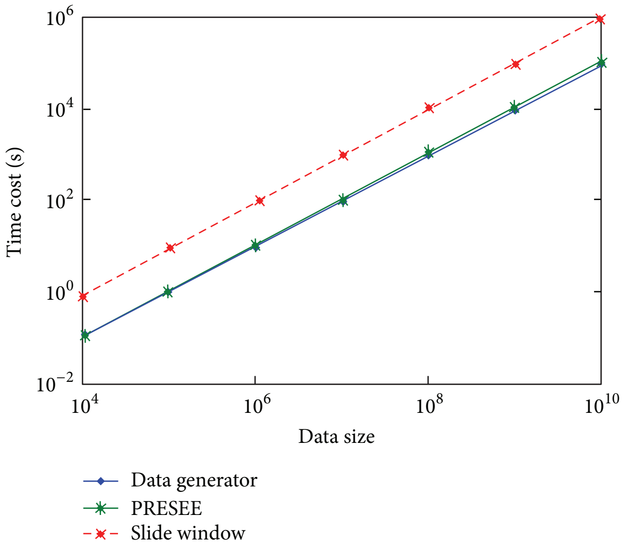


Figure 7: Efficiency and scalability of different algorithms.

algorithm, but SWBU uses BU, an $O(n \log n)$ algorithm, as the base algorithm in slide window. With larger error rate threshold set, slower SWBU would run.

Features Affecting Algorithm Efficiency. In this section, we do some experiments to explore the characteristic of our algorithm. There are two optional parameters for PRESEE: window size and batch size. Window size controls the number of points to process at once; batch size controls how many result segments are output per time. As is mentioned before, there is no necessary parameter in PRESEE, so the user can directly use default values for the two parameters. In order to see how the two parameters affect the algorithm's efficiency,

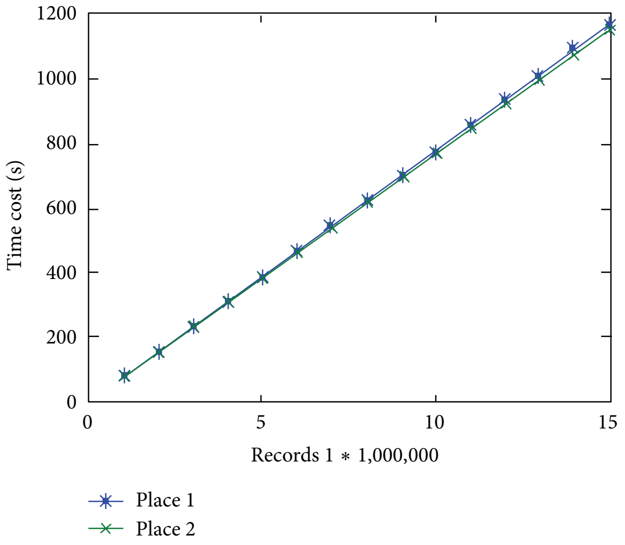


Figure 8: Efficiency of PRESEE for different datasets. Datasets are gathered from different places by ChinaFLUX.

Table 3: Relative speed of SWBU and PRESEE compared with data generated by the data generator.

Algorithm	Relative speed (t_{seg}/t_{gen})
SWBU	9.7114
PRESEE	1.1311

we run synthetic dataset number 3 for 100 times, set window size from 100 to 1000 and batch size from 1 to 10. Figure 9 shows the triple relationship among the efficiency, window size, and batch size. This figure indicates that the time cost decreases while the batch size increases, but if the batch size is too large, the efficiency of algorithm decreases. The reason is that more segments output per time can accelerate the speed of algorithm and let the slide window move faster at first. Gradually, the efficiency begins to decrease with segments size increases. There are two reasons why this phenomenon happens: firstly, the points in slide window do not have such many segments, the efficiency cannot increase forever. Secondly, the algorithm would cost extra time to identify the character points of each segment.

Compress Rate. There is no parameter to control the precision of result since MDL/MML owns the self-adaptive property. We do an experiment to see the *compress rate* (*compress rate* = *size of result*/*size of original dataset*) on different kinds of datasets. We choose ChinaFLUX datasets, IBM stock price, ICU and Dodgers datasets, and synthetic datasets 4–8 to do the experiments. The four real datasets are quite different in their appearance and they are different in cyclical/noncyclical, sharp/smooth, degree of noise, dimensions, and length. The synthetic datasets 4–8 have the same parameter on data size and range, and they are generated in the same way. The only difference is the degree of *sharpness*.

From Table 4, we can conclude that, for the same kind of data and with different *sharpness*, the compress rate is steady.

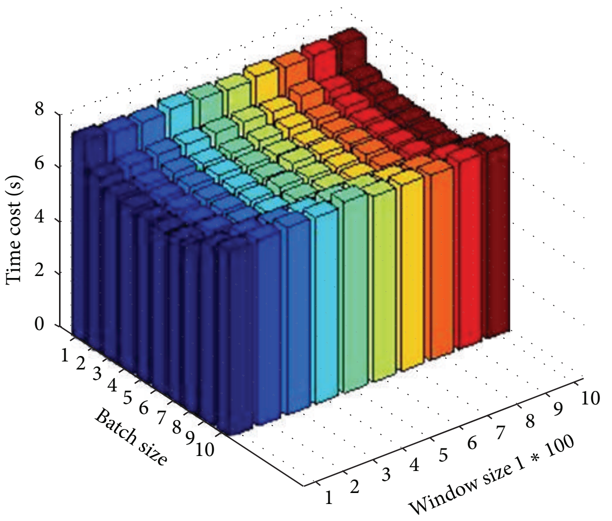


FIGURE 9: Relationship among efficiency, window size, and batch size.

TABLE 4: Compress rate of datasets.

Number	Data size	Compress rate
4	10,000,000 (10 M)	1 : 0.27548
5	10,000,000 (10 M)	1 : 0.27536
6	10,000,000 (10 M)	1 : 0.27537
7	10,000,000 (10 M)	1 : 0.27531
8	10,000,000 (10 M)	1 : 0.27536
12	IBM stock price	1 : 0.43779
13	ICU	1 : 0.48468
14	Dodgers	1 : 0.35540
15	ChinaFLUX	1 : 0.48194

7. Conclusion and Future Work

In this paper, we have proposed a new algorithm for time-series stream segmenting that is parameter free, scalable and self-adaptive. We also undertake several sets of time-series experiments on a variety of time series data types and compare them with the state-of-the art algorithms to evaluate our algorithm. The empirical results prove that PRESEE can generate a proper number of segments for time-series streams. Moreover, it can handle large dataset up to gigabytes. Finally, the parameters of PRESEE would only affect the efficiency but not the segmenting result.

In the future, we plan to design a series of time-series stream mining algorithms under the pipeline framework. Those algorithms should be able to well concatenate to the segmenting algorithm. Another direction is to design the algorithms that can do time-series mining tasks across multiple streams in real time.

Acknowledgments

This work was supported by NSF under Grant no. 31071700, the Natural Science Key Foundation of Guangxi under Grant no. 2011GXNSFD018025, and the Development Foundation of CUIT under Grant no. KYTZ201108. The authors would like to thank Dr. Yue Wang for discussing an early version of this paper.

References

[1] J. F. Roddick and M. Spiliopoulou, "A survey of temporal knowledge discovery paradigms and methods," *IEEE Transactions on Knowledge and Data Engineering*, vol. 14, no. 4, pp. 750–767, 2002.

[2] M. Brooks, Y. Yan, and D. Lemire, "Scale-based monotonicity analysis in qualitative modelling with flat segments," in *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI '05)*, pp. 400–405, 2005.

[3] C. S. Perng, H. Wang, S. R. Zhang, and D. S. Parker, "Landmarks: a new model for similarity-based pattern querying in time series databases," in *Proceedings of the 16th IEEE International Conference on Data Engineering (ICDE '00)*, pp. 33–42, March 2000.

[4] J. Friedman, "Multivariate adaptive regression splines," *Annals of Statistics*, vol. 19, pp. 1–141, 1991.

[5] Z. Luo and G. Wahba, "Hybrid adaptive splines," *Journal of the American Statistical Association*, vol. 92, no. 437, pp. 107–116, 1997.

[6] D. B. Percival and A. T. Walden, *Wavelet Methods for Time Series Analysis*, Cambridge University Press, New York, NY, USA, 2000.

[7] R. A. C. Faloutsos and A. Swami, "Efficient similarity search in sequence databases," in *Proceedings of the 4th Conference on Foundations of Data Organization and Algorithms*, pp. 69–84, 1993.

[8] V. Lavrenko, M. Schmill, D. Lawrie, P. Ogilvie, D. Jensen, and J. Allan, "Mining of concurrent text and time series," in *Proceedings of the 6th International Conference on Knowledge Discovery and Data Mining*, pp. 37–44, 2000.

[9] X. Ge and P. Smyth, "Segmental semi-Markov models for endpoint detection in plasma etching," *IEEE Transactions on Semiconductor Engineering*. In press.

[10] K. Teymourian, M. Rohde, and A. Paschke, "Knowledge-based processing of complex stock market events," in *Proceedings of the 15th International Conference on Extending Database Technology (EDBT '12)*, pp. 594–597, 2012.

[11] L. A. Tang, X. Yu, S. Kim, J. Han, C. C. Hung, and W. C. Peng, "Tru-alarm: trustworthiness analysis of sensor networks in cyber-physical systems," in *Proceedings of the 10th IEEE International Conference on Data Mining (ICDM '10)*, pp. 1079–1084, Sydney, Australia, December 2010.

[12] ICU USER GUIDE, 2012, http://userguide.icu-project.org/icudata.

[13] J. David and C. Mackay, *Information Theory, Inference & Learning Algorithms*, Cambridge University Press, New York, NY, USA, 2003.

[14] P. Grünwal, *The Minimum Description Length Principle*, MIT Press, Boston, Mass, USA, 2007.

[15] X. Lian and L. Chen, "Efficient similarity search over future stream time series," *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, no. 1, pp. 40–54, 2008.

[16] M. Kontaki, A. N. Papadopoulos, and Y. Manolopoulos, "Continuous trend-based classification of streaming time series," in

Advances in Databases and Information Systems, vol. 3631 of *Lecture Notes in Computer Science*, pp. 294–308, 2005.

[17] X. S. Wang, L. Gao, and M. Wang, "Condition evaluation for speculative systems: a streaming time series case," in *Proceedings of the 2nd Workshop on Spatio-Temporal Database Management (STDBM '04)*, pp. 65–72, 2004.

[18] P. P. Rodrigues, J. Gama, and J. P. Pedroso, "ODAC: hierarchical clustering of time series data streams," in *Proceedings of the 6th SIAM International Conference on Data Mining*, pp. 499–503, Bethesda, Md, USA, April 2006.

[19] E. Bingham, A. Gionis, N. Haiminen, H. Hiisilä, H. Mannila, and E. Terzi, "Segmentation and dimensionality reduction," in *Proceedings of the 6th SIAM International Conference on Data Mining*, pp. 372–383, April 2006.

[20] E. Terzi and P. Tsaparas, "Efficient algorithms for sequence segmentation," in *Proceedings of the 6th SIAM International Conference on Data Mining*, pp. 316–327, April 2006.

[21] D. Lemire, "A better alternative to piecewise linear time series segmentation," in *Proceedings of the 7th SIAM International Conference on Data Mining*, pp. 545–550, April 2007.

[22] J. Hunter and N. Mcintosh, "Knowledge-based event detection in complex time series data," in *Proceedings of the Joint European Conference on Artificial Intelligence in Medicine and Medical Decision Making (AIMDM '99)*, pp. 271–280, 1999.

[23] A. R. Barron and T. M. Cover, "Minimum complexity density estimation," *IEEE Transactions on Information Theory*, vol. 37, no. 4, pp. 1034–1054, 1991.

[24] J. Rissanen, "Modeling by shortest data description," *Automatica*, vol. 14, no. 5, pp. 465–471, 1978.

[25] C. S. Wallace and D. M. Boulton, "An information measure for classification," *Computer Journal*, vol. 11, no. 2, pp. 185–194, 1968.

[26] Q. Q. Lu, "An MDL approach to the climate segmentation problem," *Annals of Applied Statistics*, vol. 4, no. 1, pp. 299–319, 2010.

[27] J. G. Lee, J. Han, and K. Z. Whang, "Trajectory clustering: a partition-and-group framework," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '07)*, pp. 593–604, June 2007.

[28] K. Xu, C. Tang, C. Li, Y. Jiang, and R. Tang, "An MDL approach to efficiently discover communities in bipartite network," in *Database Systems for Advanced Applications*, vol. 5981 of *Lecture Notes in Computer Science*, pp. 595–611, 2010.

[29] C. S. Wallace and D. L. Dowe, "Minimum message length and Kolmogorov complexity," *Computer Journal*, vol. 42, no. 4, pp. 281–283, 1999.

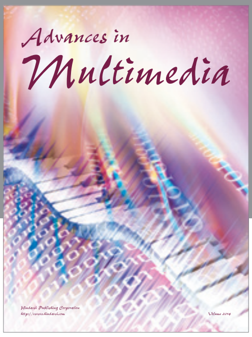
[30] P. Grunwald, I. J. Myung, and M. Pitt, *Advances in Minimum Description Length: Theory and Applications*, MIT Press, Boston, Mass, USA, 2005.

[31] IBM Stock Price, http://finance.yahoo.com.

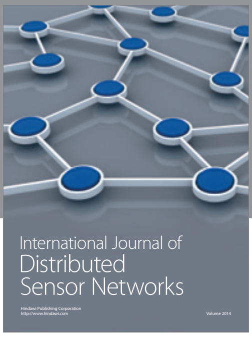
[32] A. Asuncion and D. J. Newman, *UCI Machine Learning Repository*, Irvine, University of California, School of Information and Computer Science, 2007, http://archive.ics.uci.edu/ml/datasets.

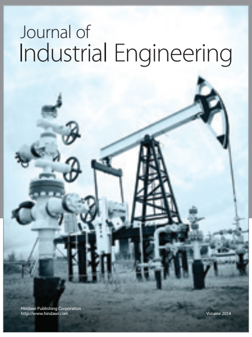
[33] G. R. Yu, X. F. Wen, X. M. Sun, B. D. Tanner, X. Lee, and J. Y. Chen, "Overview of ChinaFLUX and evaluation of its eddy covariance measurement," *Agricultural and Forest Meteorology*, vol. 137, no. 3-4, pp. 125–137, 2006.

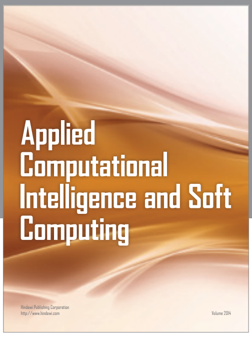
[34] J. Cheng, Y. Zhou, B. Wang, X. Wang, and J. Li, "SROS: sensor-based real-time observing system for ecological research," in *Proceedings of the International Conference on Web Information Systems and Mining (WISM '09)*, pp. 396–400, Shanghai, China, November 2009.

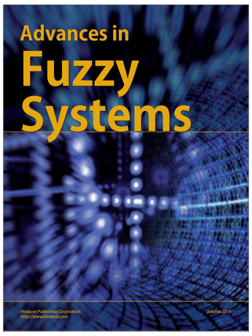

Advances in
Multimedia

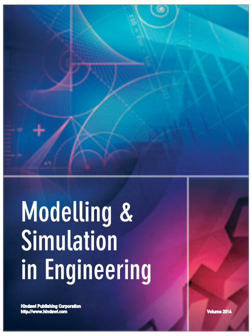
The Scientific
World Journal

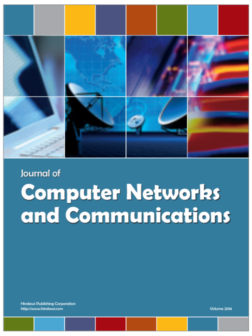

International Journal of
Distributed
Sensor Networks

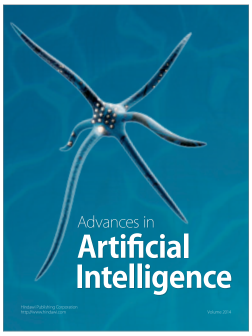

Journal of
Industrial Engineering

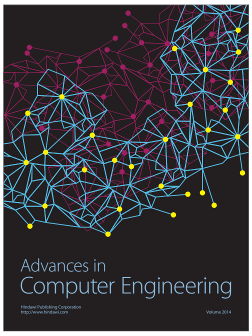

Applied
Computational
Intelligence and Soft
Computing


Advances in
Fuzzy
Systems

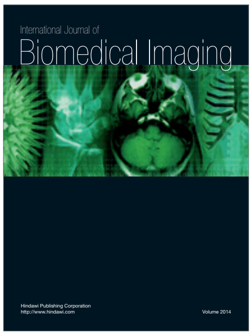

Modelling &
Simulation
in Engineering

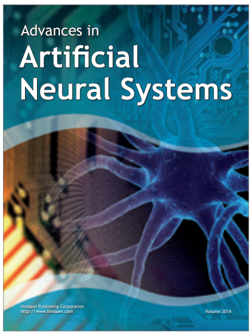

Journal of
Computer Networks
and Communications

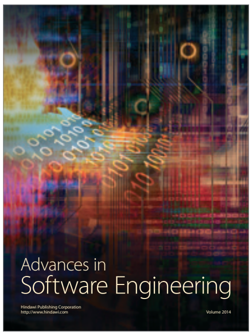

Advances in
Artificial
Intelligence


Advances in
Computer Engineering

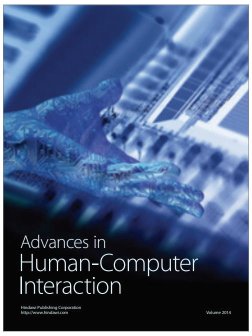
International Journal of
Computer Games
Technology

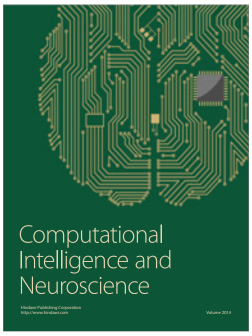

International Journal of
Biomedical Imaging


Advances in
Artificial
Neural Systems


Advances in
Software Engineering

Journal of
Robotics


Advances in
Human-Computer
Interaction


Computational
Intelligence and
Neuroscience

International Journal of
Reconfigurable
Computing

Journal of
Electrical and Computer
Engineering