Efficient Motif Discovery for Large-Scale Time Series in Healthcare

Bo Liu, *Member, IEEE*, Jianqiang Li, *Member, IEEE*, Cheng Chen, Wei Tan, *Senior Member, IEEE*, Qiang Chen, and MengChu Zhou, *Fellow, IEEE*

***Abstract*—Analyzing time series data can reveal the temporal behavior of the underlying mechanism producing the data. Time series motifs, which are similar subsequences or frequently occurring patterns, have significant meanings for researchers especially in medical domain. With the fast growth of time series data, traditional methods for motif discovery are inefficient and not applicable to large-scale data. This work proposes an efficient *M*otif *D*iscovery method for *La*rge-scale *t*ime *s*eries (MDLats). By computing standard motifs, MDLats eliminates a majority of redundant computation in the related arts and reuses existing information to the maximum. All the motif types and subsequences are generated for subsequent analysis and classification. Our system is implemented on a Hadoop platform and deployed in a hospital for clinical electrocardiography classification. The experiments on real-world healthcare data show that MDLats outperform the state-of-the-art methods even in large time series.**

***Index Terms*—Data mining, motif, pattern discovery, time series.**

I. Introduction

LARGE AMOUNTS of data are being collected from scientific experiments, business operations, and social media in the big data era [1], [2]. As a special case, thousands of research labs and commercial enterprises are generating large amounts of temporal data. For example, many research hospitals have trillions of data points of electrocardiography (ECG) data; Twitter collects 170 million temporal data every minute and serves up 200 million queries per day [3]. These temporal data are sequentially collected over time, and are called *time series*. A time series is a sequence of real numbers where each number represents a value at a point of time. Mapping to the physical world, a time series can represent network flow, exchange rates, or weather conditions over time.

Most traditional time series data mining algorithms cannot handle large-scale time series efficiently. Owing to its high time consumption, similarity search used in such algorithms becomes their bottleneck for classification, clustering, motif discovery, and anomaly detection. Time series motifs, which are similar subsequences or frequently occurring patterns, have significant meanings for researchers especially in healthcare domain. However, due to their poor scalability, existing methods on discovering motifs cannot achieve satisfactory efficiency and accuracy when facing large-scale time series data.

Motif discovery methods can basically be divided into *exact methods* and *approximate ones*. The former [4] are able to find motifs exactly, but are often inefficient when dealing with large datasets. The latter map segmentations into low dimensional words to reduce the computational complexity and execution time. For example, Liu *et al.* [5] report that it takes 5 s to find approximate motifs in a dataset with 12 500 data points, whereas exact algorithms needs about 16 s to find motifs in a 8400-point dataset. Compared with the exact algorithms, the approximate ones have much higher efficiency but lower accuracy. Moreover, additional tasks are required to determine the length of the motif when the pattern is unknown *a priori*. To resolve this problem, some studies [6], [7] utilize a scan in advance and then join short patterns to generate a longer one. When the time series are huge, these methods are inefficient.

In order to overcome the drawbacks of the present motif discovery methods, we propose a *M*otif *D*iscovery method for *La*rge-scale *t*ime *s*eries data (MDLats) by combining the advantages of both exact and approximate methods. It balances the efficiency and accuracy by computing standard motifs, based on which the final motifs, including all the similar subsequences, are discovered adaptively.

We have implemented MDLats on Hadoop [8]. Experiments on public University of California, Riverside (UCR), time series data [9] and random walks dataset [4] show that it can scale to a very large size and well outperforms competitors [4] in processing time, precision, and recall. Its real-world healthcare application on ECG classification achieves over 95% precision and recall.

II. Related Work

A. Motif Discovery

The most typical exact motif discovery method, Brute Force (BF) [10], requires a number of comparisons quadratic in the length of the time series. For each subset t of size n in time series T, BF compares it with each subset s in subsequence S to find the closest match. Its time complexity is $O(m^2)$

Manuscript received October 28, 2014; revised January 20, 2015; accepted February 19, 2015. Date of publication March 09, 2015; date of current version June 02, 2015. Paper no. TII-14-1189.

B. Liu is with the Department of Automation, Tsinghua University, Beijing 100084, China, and also with NEC Laboratories China, Beijing 100084, China (e-mail: liu_bo@nec.cn).

J. Q. Li is with the School of Software Engineering, Beijing University of Technology, Beijing 100022, China (e-mail: lijianqiang@bjut.edu.cn).

C. Chen and Q. Chen are with the Chinese Academy of Sciences, Beijing 100190, China (e-mail: chencheng@tca.iscas.ac.cn; c.qiang5882@gmail.com).

W. Tan is with the IBM T. J. Watson Research Center, Yorktown Heights, NY 10598 USA (e-mail: wtan@us.ibm.com).

M. C. Zhou is with the New Jersey Institute of Technology, Newark, NJ 07102 USA (e-mail: zhou@njit.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TII.2015.2411226

where m is the length of a time series. Mueen–Keogh (MK) [4], a tractable exact algorithm, is faster than BF by using linear ordering of data to provide useful information to guide the search for motifs.

Due to the inefficiency of exact algorithms, they are unacceptable for large time series. Many researchers have turned to approximate algorithms for motif discovery [11]–[13]. Chiu *et al.* [11] use Symbolic Aggregate approXimation (SAX) [14] to convert a time series into symbol representation and propose the first approximate algorithm for motif discovery. Using random projection (RP) [15], Buhler and Tompa give their algorithm to find motifs with very high probability even in the presence of noise. Since their work [14], [15], more than a dozen of approximate algorithms have been proposed. Their complexity is mostly $O(m)$ or *O(m*log*m)* with high constant factors, and accuracy is lower than that of exact algorithms.

Compared to existing methods, our proposed method MDLats has the following advantages.

1) It improves the efficiency for motif discovery while retaining accuracy. By computing standard motifs, it can reduce majority of redundant computation, and reuse existing information to the maximum by exploiting the relation between existing data and newly arrived ones.
2) It can generate all the motifs, including not only the most similar pairs of subsequences as commonly done in existing methods but also all the similar subsequences, which provide additional information for time series analysis.
3) It is adaptive to different length of motifs, and can avoid a full scan in the preprocessing. It is extremely efficient when the time series are regular and the fluctuations in them are sparse. Generally, both ECG and network flow data enjoy this feature.
4) It is scalable and can be deployed in Hadoop for parallel computing.

B. Hadoop

Our proposed method is implemented on a MapReduce and Hadoop platform [8], [16]. Hadoop [8] supports data-intensive distributed applications on large clusters, which is an open-source Java implementation of MapReduce. It uses a distributed file system, manages the execution of data transfer, and can efficiently run and monitor MapReduce applications. MapReduce is a programming model for parallel and distributed processing of batch jobs [16]. By writing *Map* and *Reduce* functions, a MapReduce programming model provides a convenient way for programmers to process large datasets. There are increasing machine learning tools using Hadoop, such as Mahout [17] and Hadoop-ML [18]. Some researchers [19]–[21] build their algorithms on Hadoop for big data analysis.

III. Preliminary Definitions

This section gives definitions used throughout this paper.

Definition 1: A *time series* T is an ordered list $T = \langle t_1, \ldots, t_m \rangle$ of real-valued variables, where m is its length.

While the source data can be a long time series, most researchers confine their interests to its subsections, which are called *subsequences*.

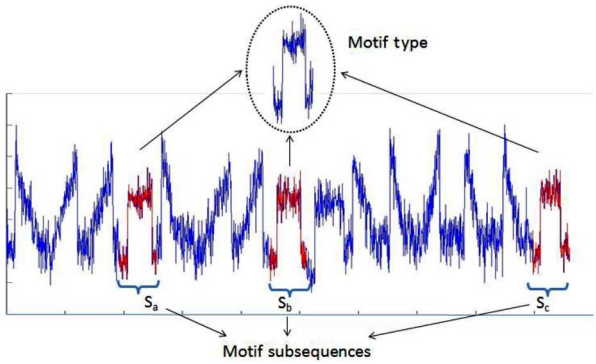

Fig. 1. Motif type and three motif subsequences S_a, S_b, and S_c.

Definition 2: Given a time series T of length m, a *subsequence* $T_{i,k}$ of T is a time series of length $k < m$, which starts from position i, i.e., $T_{i,k} = \langle t_i, t_{i+1}, \ldots, t_{i+k-1} \rangle$, $1 \leq i \leq m - k + 1$.

The distance between two subsequences is often computed by Euclidean distance (ED).

Definition 3: Given two time series (or subsequences) $Q = \langle q_1, q_2, \ldots, q_n \rangle$ and $P = \langle p_1, p_2, \ldots, p_n \rangle$ both of length n, the ED between them is defined as

$$D(Q, P) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

If a given subsequence is similar to another subsequence, we claim a *match* as defined below.

Definition 4: Given a time series T containing two subsequences $S_{i,k}$ and $S_{j,k}$ with length k, if their distance $D(S_{i,k}, S_{j,k})$ is smaller than a predefined threshold r, then $S_{j,k}$ is a *matching* subsequence of $S_{i,k}$. If $S_{i,k}$ and $S_{j,k}$ start at significantly different positions, they are called a *nontrivial match*. We say their starting positions, i.e., i and j, are significantly different if there exists x, such that $i < x < j$ and the distance $D(S_{i,k}, S_{x,k})$ is larger than r.

In a time series T, some subsequences appear many times or exist in many matching segmentations. Let $\Gamma(S_{i,k})$ represent the number of nontrivial matches to $S_{i,k}$. If there is none, $\Gamma(S_{i,k}) = 0$.

Definition 5: Motif: Given k (subsequence length) and r (predefined distance threshold), the most significant *motif* of length k in T is the subsequence $S_{i,k}$ if $\forall\, S_{j,k}$ (subsequence in T), $\Gamma(S_{j,k}) \leq \Gamma(S_{i,k})$.

Clearly, the most significant motif is not necessarily unique. Since we pay more attention to the pattern of the motif and the subsequences that match the pattern, we further define a *motif type* and *motif subsequences*.

Definition 6: Motif type is the pattern of a motif, which is presented as $M_T = \langle v_1, v_2, \ldots, v_k \rangle$, where v denotes a set of values of data points and k is the motif length.

Definition 7: Motif subsequence is a subsequence that matches the motif type within a time series T. A motif type often has multiple motif subsequences.

Taking the CBF dataset [9] as an example, we detect three similar subsequences as a motif, as seen in Fig. 1.

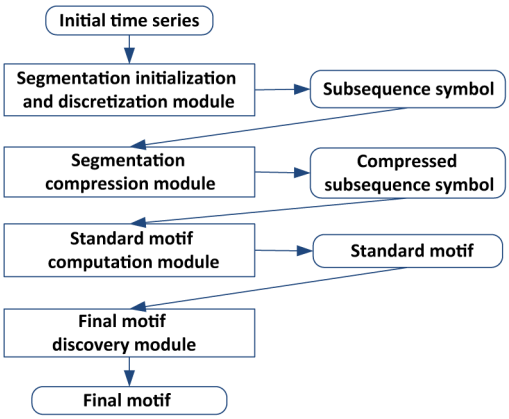

Fig. 2. Main steps of MDLats.

IV. Proposed Method

A. Intuition Behind MDLats

To find the motifs efficiently, approximate algorithms usually require a discrete representation of the time series whose dimensionality is reduced upon a lower bounding distance measure. One of their drawbacks is that some useful information may be lost during the dimensionality reduction and discretization, and the resulting patterns may be incomplete. They identify only each pair of matching patterns but fail to summarize all the occurrences of a pattern for general cases.

In this paper, we present a method called MDLats. It summarizes an average pattern based on the occurrences of the pattern, which we denote as "Standard Motifs." To find all the possible patterns in a time series, we first compute some candidates by an improved approximate method based on the RP algorithm [11]. Then, we obtain the standard motifs by computing the ED of the original data between two candidates. Since the number of the resulting candidates is much smaller than that of original subsequences, this process takes little time. The observation is that if two subsequences are close in Euclidean distance, they should also be close to the standard motifs. By comparing with standard motifs, all the motif types and subsequences are obtained, which can be used for subsequent analysis, such as classification, clustering, pattern analysis, and anomaly detection.

Before its detailed explanation, Fig. 2 gives an overview of our method. The input of the system is an initial time series. The Segmentation Initialization and Discretization Module preprocesses it to obtain subsequence symbols by using SAX [14] and Piecewise Aggregate Approximation (PAA) [22] methods. The Segmentation Compression Module compresses subsequence symbols by abstracting repetitive symbols. The Standard Motif Computation Module then calculates standard motifs for the following Final Motif Discovery Module, which in turn generates a set of final motifs that have similar patterns.

B. Segmentation Initialization and Discretization

In order to preprocess the initial time series data, the first step in our procedure is to discretize the series. It provides a lower dimensional presentation that reduces the effect of the noise in the raw time series data and at the same time preserves its main properties. It also gives a natural string presentation that will be used in the subsequent step by the RP algorithm. We choose SAX and PAA for our discretization step.

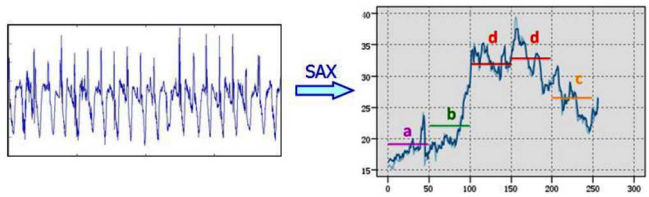

Fig. 3. SAX.

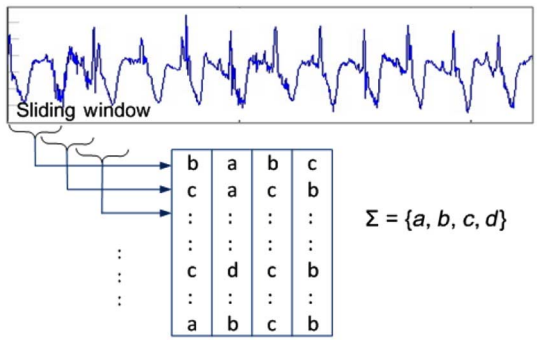

Fig. 4. Symbolic representations of time series.

SAX accepts as input parameters the time series $T = \langle t_1, t_2, \ldots, t_m \rangle$ that we want to discretize, the desired length w of the symbolic representation and the size $|\Sigma|$ of the alphabet to be used. This lower dimensional representation, known as PAA, speeds up the computation procedure. Once all the variables are computed, they are quantized into $|\Sigma|$ intervals. Finally, SAX assigns the same alphabet to all the variables that belong to the same interval. Fig. 3 illustrates the discretization of a time series by using SAX. A small segmentation of original time series can be represented as five letters "abddc" after SAX. The size of the alphabet is 4 and $\Sigma = \{a,\ b,\ c,\ d\}$.

Next, we should determine the length of the motif (w) in advance. An inappropriate w value may significantly affect the results of the random projection. If w is set to be too small, the algorithm can only identify the subsequences of a hidden pattern. If w is set to be too large, no patterns can be discovered. When there are several patterns with different lengths in the time series, the matching task becomes even more difficult.

Our strategy is to set a relatively small w value to identify some relatively short patterns first. Thus, many candidate subsequences of motifs can be generated from the random projection. Then, we "concatenate" the discovered patterns to generate the final motifs. Since all the motifs are included in the candidates, we can discover the whole motifs with different lengths from these short patterns abstracted from the collision matrix during the random projection. Given w, the symbolic representations of the plotted time series are stored in tables as illustrated in Fig. 4.

C. Segmentation Compression

This step is to eliminate some redundant subsequence in the motif discovery by compressing the data of discretized

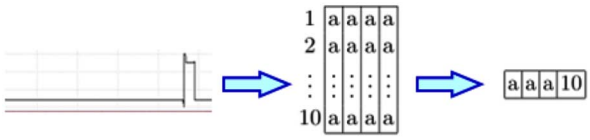

Fig. 5. Segmentation compression.

subsequences. In large time series, the subsequences sometimes appear to be regular and the fluctuations are sparse, for instance, ECG data or network flow data. We denote these subsequences as trivial segmentations, when they appear to be similar and repeat in the phase of the trivial segmentations in the symbol representation. This procedure can be done by a fast scan over all the subsequences of the symbols. If the same symbol, e.g., "aaaa," appears repeatedly, the interval is defined as a trivial one. Then, we compress them into one symbolic representation with the length of the interval as shown in Fig. 5.

D. Standard Motif Computation

In this step, we start to project the similarity of each pair of subsequences into a collision matrix. Projection applies a set of hash functions to all the subsequences of length w. Each function splits the set of substrings into a number of classes of equivalence. When all the functions are applied, if in some of those classes there are more than a predefined number of strings hashed, then with a very high probability these strings correspond to a planted motif. Once we have the symbolized sequences, we need to find out which of them correspond to approximate motifs.

Fig. 6 shows an example of a collision matrix. Suppose that we have a symbolic table including five subsequences: S1 = babc, S2 = cacb, S3 = cdcb, S4 = babc, and S5 = abcb. In the first iteration of the random projection, the first and third dimensions (shaded) are selected as projecting dimensions. We find that S1 and S4 are identical, and S2 and S3 are identical. Then, the corresponding positions in the collision matrix are increased with one. Note that their initial values are 0. In the second iteration, the second and fourth dimensions (shaded) are selected, and S1 and S4 are increased with one. Since a subsequence is the same as itself, we ignore the comparison with itself and let the values on the diagonal retain 1 for more efficient processing. Note that the selection of dimensions and iteration count are flexible and user-defined. Since the collision matrix is symmetrical, we need to compute half of its values only.

The described procedure is repeated m times and in each iteration, new random projecting positions are selected and the counters in the collision matrix are being set or increased. After the last iteration, a threshold s is applied, which filters all positions in the matrix that have a counter larger than or equal to s. If the algorithm correctly identifies the matching pairs of the subsequences, some dot plots appear on the corresponding locations of the collision matrix. Finally, we obtain a sparse collision matrix with a few dot plots, as given in Fig. 7.

For each dot plot in the matrix, we can determine those neighboring subsequences that are close enough in the original time series. However, these subsequences are short motifs

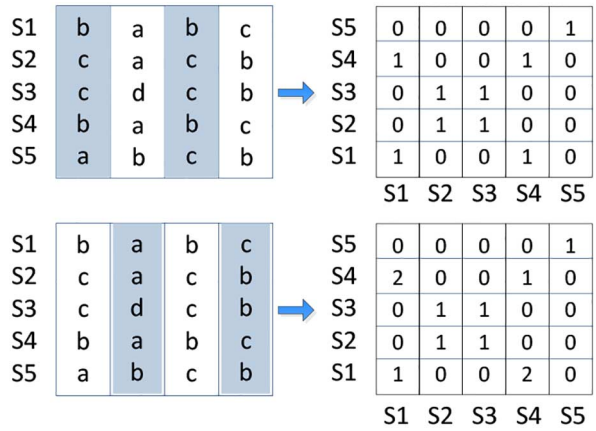

Fig. 6. Collision matrix.

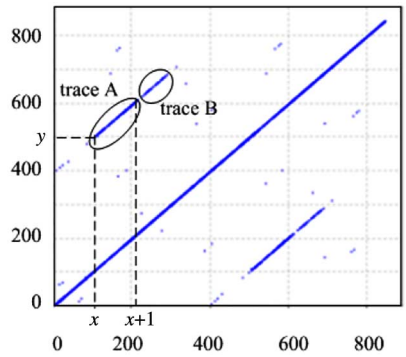

Fig. 7. Dot plots and traces.

rather than complete ones since there exist submotifs when the length of the motif is small. Discovering the complete motifs can be more interesting than just discovering the submotifs. To extract them, we take advantage of the length of plots in the matrix. For a trace with the start coordinate (x, y) with length l, we concatenate those neighboring motifs that are close enough by connecting them along a line segment. First, we sort the dots by x. Then, we start from position a whose coordinate is $(x + l, y + l)$ and search its width d window. If dot b with coordinate (x', y') in the search window is close enough to a within this slope, we put b to the end of the trace to obtain a new line with the start coordinate (x, y) with length $x' - x + 1$. After searching all the dots, the resulting subsequences are motifs. Algorithm 1 describes how to find possible motifs from a collision matrix.

Algorithm 1. *find_possible_motif* (*setM*, S, d, b) (Find possible motifs from a collision matrix)

Input: A set of plots generated by a collision matrix $setM = \{(x_i, y_i, L_i)\}$, where x_i and y_i are the initial x-coordinate and y-coordinate of the nonzero trace in the matrix and L_i denotes the length of the trace; a set of original subsequences with fixed length $S = \{s_1, s_2, \ldots\ldots, s_m\}$; the search window width d; and ED lower bound b.

Output: A set of motif candidates $M = \{M_i\} = \{(s_{i1}, s_{i2}, \ldots\ldots, s_{ik})\}$, where each M_i denotes a motif type and s denotes a subsequence in the time series.

BEGIN
For each trace (x, y, L) in *setM* // Decide the position of the segmentations by the coordinate of the plot.
If exist another trace (x', y', L') satisfy $(x' - x - L) < d$ // x' is in the search window.
If $(y - x)/(y' - x') \in [0.9,\ 1.1]$ // traces (x, y, L) and (x', y', L') are in the same slope, [0.9, 1.1] is the allowed range of gradient.
$Sum = 0$;
For $i = 1$ to $L + d + L'$
Sum = Sum + $|s_{x+i} - s_{y+i}|$;
If $Sum > b$ // If the ED among subsequences is larger than the bound
Replace trace (x, y, L) with trace(x, y, i) in *setM*;
Break; // abort the innermost *For* loop
End If
End For
End If
End If
End For
Group *setM* by the value of x; // the traces with same x are grouped to a subset
$setM = \{A_1, \ldots\ldots, A_n\}$; // $A_i = \{(x_{i1},\ y_{i1},\ L_{i1}), \ldots, (x_{ip},\ y_{ip},\ L_{ip})\}$ in which $x_{i1} = x_{i2} = \cdots = x_{ip}$
For each A_i in *setM*
For each trace (x_{ip}, y_{ip}, L_{ip}) in A_i
Abstract the two subsequences corresponding to the trace and record them in M_i; // each trace corresponds to two subsequences from x-coordinate and y-coordinate.
End For
End For
$M = \{M_i\} = \{(s_{i1}, s_{i2},\ \ldots\ldots,\ s_{ik})\}$;
Return M
END

Since the matrix threshold value is large, the motif subsequences found from Algorithm 1 are close enough in space. Based on this, we believe that these motif candidates are reliable. Therefore, we deal with them as a training set. To confirm a most standard motif and eliminate the noise, we construct a centroid classifier to compute a center and deviation of motif groups. The details of the method are described in Algorithm 2.

Algorithm 2. *find_standard_motif* (S) (Generate standard motifs by clustering from a set of possible subsequences)

Input: A set of motif candidates $M = (s_1, \ldots, s_k)$, where $s_1, \ldots, s_k$ represent *k* subsequences that belong to a common motif type.
Output: A set of standard motifs $M' = \{s_1', \ldots, s_k'\}$, and its center s and deviation δ presented as (s, δ).
BEGIN
$M' = (s_1', \ldots, s_k') \leftarrow Cluster(M, 1)$; // *Cluster* is a k-means-based clustering algorithm, and outputs one cluster M'
$s \leftarrow s_1'$; // The first element in M is the center of this cluster
$\delta = 0$;
For $i = 1$; $i < k + 1$; $i{+}{+}$
$\delta = \delta + (s - s_i')^2$;
End For
$\delta = \sqrt{\delta/k}$; // Compute the deviation, k is the number of subsequences in M
Return M', (s, δ)
END

E. Final Motif Discovery

With the obtained information from the last step, the standard motif and its deviation, we can efficiently discover more motifs with the same motif type.

First, we gradually decrease the threshold value of the matrix. By choosing a value r smaller than the initial matrix threshold e, more plots appear in the new matrix. The selection of r depends on a specific scenario and data range and, in general, we set $r = 0.8e$. Second, we extract the subsequences that can possibly be motifs from the plots. Third, we decide whether they are motifs by computing the distance between the extracted subsequence and the corresponding standard motif. If their ED is lower than δ, the subsequence belongs to this motif type. The above steps are repeated till no more new motifs are found. They are described in Algorithm 3.

Algorithm 3. *find_final_motif* (*SD,* T, e) (Find final motifs based on standard motifs)

Input: The means and deviation of the standard motifs $SD = \{(s_1,\ \delta_1),\ \ldots,\ (s_m,\ \delta_m)\}$, the symbolic representation table T, and the initial matrix threshold e.
Output: Final motifs M.
BEGIN
Do{
Choose a value r smaller than e;
Use the threshold r to build the collision matrix;
Find each plot (x_i, y_i, L_i) in the collision matrix
$s' \leftarrow$ the x-coordinate segmentation of the plot;
If exists a standard motif s_i in the area (x_i, $x_i + L_i$)
If $|s' - s_i| < \delta_i$
s' is a valid motif and add s' to M_i;
$e \leftarrow r$;
}
While {no new motifs are found}
Return $M = \{M_i\}$
END

V. Implementation and Experiment

A. System Architecture

We implement MDLats on the Hadoop platform because it can scale out to many machines, and make computation close to where data are. Fig. 8 presents the architecture of our system. First, the input time series are mapped to one or more computing nodes; and then in each node, a set of *reduce* functions are conducted to implement the core modules of MDLats, including segmentation initialization, discretization, compression, standard motif computation, and final motif discovery. Lastly, the final motifs obtained in each node are mapped to a common node (e.g., node p), and in the *reduce* phase, they are merged and output to users as final motifs.

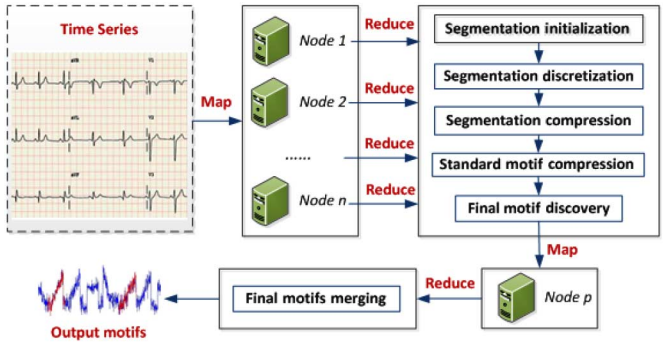

Fig. 8. System architecture.

TABLE I
UCR DATASETS

Dataset	Total length (data points)	# of time series	Length of each time series	# of classes
ECG	19 200	200	96	2
CBF	119 040	930	128	3
Gun-point	30 000	200	150	2

B. Performance Evaluation

Our experiments are divided into three stages. First, we use a small or middle-sized datasets to validate the effectiveness of our method in a clear way. UCR time series [9] is selected for this purpose. It includes 47 public datasets for time series classification and clustering. Then, a larger dataset, Random Walks [4], is used to validate the scalable performance. Finally, our method is applied for real-world ECG classification for a Chinese hospital, which demonstrates its utility in healthcare. Note that although we take healthcare as an application, MDLats is a general method for motif discovery and can be applied to other domains that have time series data.

1) Dataset: Of the 47 datasets in UCR time series, we use three for first-stage experiments, which are ECG, CBF, and Gun-Point. Their features are shown in Table I. The ECG dataset [9] contains 200 physiological time series, such as respiratory recordings, heart beats, and brain waves. The CBF dataset [9] is created by the cylinder-bell-funnel synthetic approach, including 930 time series. The Gun-Point dataset has 200 time series, each with 150 data points.

2) Execution Time: To validate our method, we compare it with three typical baseline methods, including two exact motif discovery methods, BF and MK, and one approximate method, RP.

BF [10] maintains a running minimum best-so-far and updates it whenever the algorithm finds a pair of time series having a smaller distance between them. Its complexity is $O(m^2)$ where m is the length of the time series. MK [4] is a more tractable exact algorithm that is faster than BF. RP [15] can find motifs with high probability, but this also depends on its setting of matrix threshold. Generally, it is faster than exact algorithms.

We use them for the comparison with MDLats on ECG dataset. Each algorithm is run 10 times and the average execution time is recorded in Table II. In order to clearly show the performance of RP and MDLats, the matrix threshold of RP is set to $e = 40$ and $e = 80$, respectively, and for MDLats, the high matrix threshold is $e_{\mathrm{H}} = 80$ and low matrix threshold is $e_{\mathrm{L}} = 40$. It is clear that MDLats is significantly faster than MK and BF, i.e., its execution time is only 11% of MK and 1.5% of BF. This is because it reduces the number of comparisons by computing standard motifs at the early stage. Moreover, MK and BF only output the most similar subsequences, while MDLats generates all the similar subsequences for each motif type.

TABLE II
EXECUTION TIME ON ECG DATASET

# of times	Execution time (ms)				
	MDLats ($e_H = 80$, $e_L = 40$)	MK	BF	RP ($e = 40$)	RP ($e = 80$)
1	702	6124	40 098	783	680
2	670	6843	41 522	765	678
3	671	6057	49 172	770	669
4	701	6280	48 074	778	692
5	676	8081	48 270	786	670
6	688	5372	48 399	762	674
7	678	6751	47 832	789	671
8	697	5306	48 407	781	687
9	712	5991	48 259	792	690
10	677	5623	48 069	776	660
Average	687.2	6242.8	46810.2	778.2	677.1

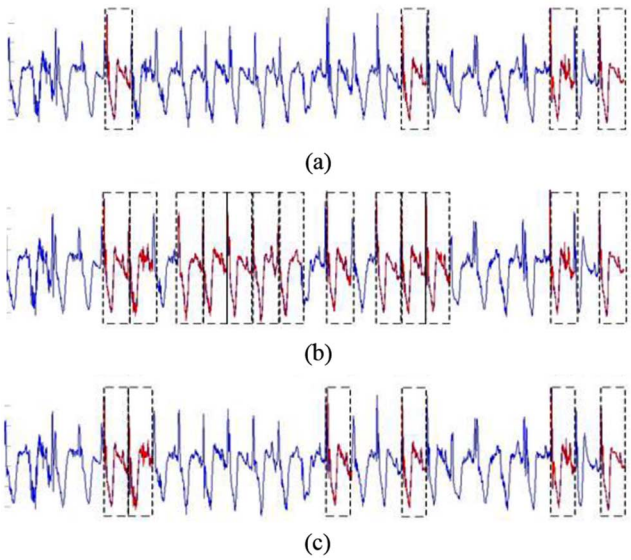

Fig. 9. Discovered motifs on ECG dataset. (a) RP, $e = 80$, 4 motifs discovered. (b) RP, $e = 40$, 13 motifs discovered. (c) MDLats, $e_H = 80$, $e_L = 40$, 6 motifs discovered.

The execution time of MDLats is smaller than that of RP ($e = 40$), and slightly longer than that of RP ($e = 80$). But looking into the motifs, we can find that MDLats better balances the accuracy and number of motifs discovered than RP does. Taking a sequence in ECG dataset as an example, when we set the matrix threshold of RP as $e = 80$, four motifs are discovered as pattern 1 [Fig. 9(a)]. These four motifs are very similar and marked in dashed boxes. Then, we decrease the matrix threshold to $e = 40$, 13 motifs are found in pattern 1

TABLE III
PRECISION AND RECALL ON CBF AND GUN-POINT DATASETS

	MDLats	RP			
CBF	$e_h = 90, e_l = 70$	$e = 100$	$e = 90$	$e = 80$	$e = 70$
Precision	0.92	0.96	0.92	0.85	0.83
Recall	0.83	0.61	0.78	0.82	0.87
F-measure	**0.873**	0.746	0.844	0.835	0.85
Gun-Point	$e_h = 110, e_l = 80$	$e = 110$	$e = 100$	$e = 90$	$e = 80$
Precision	0.92	0.92	0.93	0.83	0.85
Recall	0.9	0.84	0.86	0.86	0.89
F-measure	**0.91**	0.878	0.894	0.845	0.87

[Fig. 9(b)], but some of them are not very similar as seen in human eyes. Finally, we use MDLats, and set the high matrix threshold as $e_{\mathrm{H}} = 80$ and low matrix threshold as $e_{\mathrm{L}} = 40$; six motifs are discovered [Fig. 9(c)]. We can see that MDLats can identify more motifs than $e = 80$ and achieve higher accuracy than $e = 40$.

3) Precision and Recall: Then, we use precision and recall to evaluate the performance of MDLats. Precision is the fraction of discovered motifs that are correct motifs. Assume that we have discovered m types of motifs $\{M_1, M_2, \ldots, M_m\}$, each M_i contains n_i motifs that have the same pattern. $N(M_i)$ is the number of the motifs that are judged as correct. Recall is the fraction of all correct motifs that are successfully discovered. *F*-measure is the harmonic mean of precision and recall

$$\text{precision} = \frac{N(M_1) + N(M_2) + \cdots + N(M_m)}{n_1 + n_2 + \cdots + n_m}$$

$$\text{recall} = \frac{N(M_1) + N(M_2) + \cdots + N(M_m)}{\#\text{ of all the correct motifs}}$$

$$F\text{-measure} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}.$$

BF and MK can only find the most similar pair of motifs, while MDLats generates all the similar subsequences for each motif type, it is infeasible to compare our method with them in terms of precision and recall. Accordingly, we compare the precision, recall, and *F*-measure with RP on CBF and gun-point datasets. Results in Table III show that, with RP, decreasing matrix threshold e can result in lower precision and higher recall. MDLats achieves better tradeoff between precision and recall. It has the highest *F*-measure in both datasets.

4) Large-Scale Time Series: Since the size of UCR datasets is small, we use random walks datasets [4] for experiments on large-scale time series. We produce four sets of random walks of different sizes, containing 1.1, 10, 20, and 100 millions of data points, all of length 1024. We run MDLats 10 times on each of these datasets and take the average execution time. As seen in Table IV, MDLats outperforms MK, BF, and RP significantly on the first three datasets. Its total time is only 33.8% of MK, 1.6% of BF, and 80% of RP. For the fourth dataset with 100 million data points, all four methods fail to obtain the result due to insufficient computation capability of a single node.

To validate the scalability of MDLats, we increase the number of nodes from 1 to 8 and report the time in Fig. 10. Hadoop

TABLE IV
EXECUTION TIME (ONE NODE)

Dataset	Length (millions of points)	Execution time (ms) MDLats	MK	BF	RP
dataset1	1.1	**531**	1404	29 437	657
dataset2	10	**4162**	12 194	281800	5279
dataset3	20	**7952**	23 861	559 413	9885
dataset4	100	N/A	N/A	N/A	N/A
Total		**12 645**	37 459	870 650	15 821

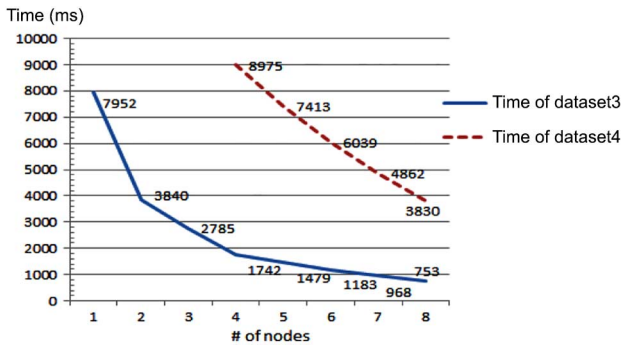

Fig. 10. Execution time with different nodes.

1.2.1 is deployed on 8 computing nodes, each with 2 cores, 3.75 GB memory, and 410 GB hard disk. When more than three nodes are deployed for parallel computation, dataset4 can be processed by MDLats. Clearly, adding more nodes can speed up the procedure. The time on dataset3 is reduced from 7952 ms on one node to 753 ms on 8 nodes, and the time on dataset4 also decreases from 8975 ms on 4 nodes to 3830 ms on 8 nodes.

5) Classification: In this section, we introduce the application of MDLats for time series classification. The Starlight Curve dataset in UCR [9] consists of 9236 starlight curve time series of length 1024. It is divided into 1000 time series for training and 8236 for testing. Based on the training set, MDLats generates standard motifs for three classes. Then for each testing time series, the discovered motifs are compared with standard ones, and this time series is assigned to the class with the highest similarity.

Keogh *et al.* [23] proposed a shapelets-based method which is the latest one with the highest accuracy to our best knowledge. They reported the average accuracy on starlight curve dataset as 93.68%, while the accuracy of the one nearest neighbor algorithm using the ED and DTW is 84.9% and 90.5%, respectively. MDLats achieves an average accuracy of 90.7% from 20 runs, slightly lower than that of Shapelets. Note that Shapelets are computed via entropy and information gain and are designed to classify unlabeled time series. MDLats is a more general method for efficient motif discovery, and that's why its accuracy is not as high as that of Shapelets.

Then, we validate MDLats on real-world healthcare data. Collaborating with a Chinese hospital, we aim to facilitate the classification of ECG data. The hospital generates 240 ECGs per day on average. Traditionally, specific doctors are responsible for classifying them into normal and abnormal ones, which

TABLE V
ECG CLASSIFICATION RESULTS

	Normal ECG	Abnormal ECG
Detected normal ECG	1505	51
Detected abnormal motifs	73	343
Unclassified	22	6

is time-consuming and limited to doctors' professional competence. Abnormal ECG indicates some diseases of patients who should be further examined.

We help the doctors classify clinical ECG data automatically. The training set contains 600 ECGs from 600 patients, in which 400 are normal ones and 200 are abnormal ones. In the training phase, if a motif type exists in both normal and abnormal ECGs, we assign it to the abnormal category since the negative examples are much fewer than the positive ones. The testing set consists of 2000 ECGs from 1850 patients, including 1600 normal ones and 400 abnormal ones. For each time series, we compare it with abnormal standard motifs first; if there exist similar patterns, this sample is classified to abnormal ECG. Otherwise, we compare it with normal standard motifs; if the number of similar patterns is more than a predefined threshold, this sample is classified as normal ECG. If it belongs to neither normal nor abnormal ECG, this sample is marked as unclassified and requires manual classification. This rule is to ensure the correctness of final judgment since the influence of "false negative" is lighter than "false positive" in the medical domain considering the health of patients.

The classification results are shown in Table V. The precision, recall, and F-measure are 96.7%, 95.4%, and 96%, respectively. Twenty-eight ECGs are not classified and sent to doctors for manual classification, since no motifs are found in them. If we consider unclassified ECG as false ones, the precision, recall, and F-measure are 96.4%, 94.1%, and 94.1%, respectively. A production-level system is deployed on Hadoop 1.2.1 with four computing nodes. During the operation in the past year, the average precision was 95.8%.

VI. CONCLUSION AND FUTURE WORK

This paper presents an algorithm and system for motif discovery in large-scale time series called MDLats. It combines the advantages of approximate methods and exact ones, and exploits the RP algorithm and the ED to find the motifs efficiently and accurately. A production-level system is implemented on a Hadoop platform. The experimental results and real-world healthcare application on ECG classification validate the effectiveness of MDLats. In the future, we will validate MDLats on more benchmark studies, and apply it to other application domains such as atmospheric pollution, social network, and logistics optimization.

REFERENCES

[1] F. Shifeng *et al.*, "An integrated system for regional environmental monitoring and management based on Internet of Things," *IEEE Trans. Ind. Informat.*, vol. 10, no. 2, pp. 1596–1605, May 2014.

[2] X. Luo *et al.*, "An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems," *IEEE Trans. Ind. Informat.*, vol. 10, no. 2, pp. 1273–1284, May 2014.

[3] B. Lorica. (2013). *How Twitter Monitors Millions of Time-Series* [Online]. Available: http://strata.oreilly.com/2013/09/how-twitter-monitors-millions-of-time-series.html

[4] A. Mueen *et al.*, "Exact discovery of time series motifs," in *Proc. SIAM Int. Conf. Data Min.*, 2009, pp. 473–484.

[5] Z. Liu *et al.*, "Locating motifs in time-series data," *Adv. Knowl. Discovery Data Mining*, vol. 3518, pp. 343–353, 2005.

[6] D. Yankov *et al.*, "Detecting time series motifs under uniform scaling," in *Proc. 13th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD'07)*, New York, NY, USA, 2007, pp. 844–853.

[7] B. Chiu, E. Keogh, and S. Lonardi, "Probabilistic discovery of time series motifs," in *Proc. 9th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2003, pp. 493–498.

[8] Hadoop. (2015). *Apache* [Online]. Available: http://hadoop.apache.org/

[9] E. Keogh *et al.* (2011). *The UCR Time Series Classification/Clustering Homepage* [Online]. Available: www.cs.ucr.edu/~eamonn/time_series_data/

[10] J. L. E. K. S. Lonardi and P. Patel, "Finding motifs in time series," in *Proc. 8th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Edmonton, AB, Canada, 2002, pp. 53–68.

[11] B. Chiu, E. Keogh, and S. Lonardi, "Probabilistic discovery of time series motifs," in *Proc. 9th Int. Conf. Knowl. Discovery Data Mining*, Washington, DC, USA, 2003, pp. 493–498.

[12] Y. Tanaka, K. Iwamoto, and K. Uehara, "Discovery of time-series motif from multi-dimensional data based on MDL principle," *Mach. Learn.*, vol. 58, no. 2–3, pp. 269–300, 2005.

[13] H. Tang and S. S. Liao, "Discovering original motifs with different lengths from time series," *Knowl. Based Syst.*, vol. 21, pp. 666–671, 2008.

[14] J. Lin *et al.*, "A symbolic representation of time series, with implications for streaming algorithms," in *Proc. 8th ACM SIGMOD Workshop Res. Issues Data Mining Knowl. Discovery*, 2003, pp. 2–11.

[15] J. Buhler and M. Tompa, "Finding motifs using random projections," *J. Comput. Biol.*, vol. 9, no. 2, pp. 225–242, 2002.

[16] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[17] Apache. (2015). *Mahout* [Online]. Available: https://mahout.apache.org/

[18] A. Ghoting and E. Pednault, "Hadoop-ML: An infrastructure for the rapid implementation of parallel reusable analytics," in *Proc. Large-Scale Mach. Learn. Parall. Massive Data Sets Workshop (NIPS'09)*, 2009, pp. 38–48.

[19] D. Wegener *et al.*, "Toolkit-based high-performance data mining of large data on mapreduce clusters," in *Proc. Int. Conf. Data Mining Workshops (ICDMW'09)*, 2009, pp. 296–30.

[20] C. Chu *et al.*, "Map-reduce for machine learning on multicore," *Adv. Neural Inf. Process. Syst.*, vol. 19, pp. 281–288, 2007.

[21] J. Wang *et al.*, "A scalable data science workflow approach for big data Bayesian network learning," in *Proc. Int. Symp. Big Data Comput.*, to be published.

[22] E. Keogh *et al.*, "Dimensionality reduction for fast similarity search in large time series databases," *J. Knowl. Inf. Syst.*, vol. 3, pp. 263–286, 2001.

[23] E. J. Keogh and T. Rakthanmanon, "Fast Shapelets: A scalable algorithm for discovering time series Shapelets," in *Proc. SDM*, 2013, pp. 668–676.

Authors' photographs and biographies not available at the time of publication.