Limited-length Suffix-array-based Method for Variable-length Motif Discovery in Time Series

Le Sun[1], Jinyuan He[2], Jiangang Ma[2], Hai Dong[3], Yanchun Zhang[2]

[1] School of Computer and Software, Nanjing University of Information Science and Technology, China
[2] Institute for Sustainable Industries and Livable Cities, Victoria University, Australia
[3] School of Science, RMIT University, Australia
sunle2009@gmail.com, jinyuan.he@live.vu.edu.au, Jiangang.Ma@vu.edu.au, hai.dong@rmit.edu.au, Yanchun.Zhang@vu.edu.au

Abstract

In this paper, we explore two key problems in time series motif discovery: releasing the constraints of trivial matching between subsequences with different lengths and improving the time and space efficiency. The purpose of avoiding trivial matching is to avoid too much repetition between subsequences in calculating their similarities. We describe a limited-length enhanced suffix array based framework (LiSAM) to resolve the two problems. Experimental results on Electrocardiogram signals indicate the accuracy of LiSAM on finding motifs with different lengths.

Keywords: Time series, Motif discovery, Enhanced suffix array, ECG

1 Introduction

Motifs of a time series are the frequently-occurred and approximately similar subsequences that can summarize the features of the time series [1]. Motifs have been applied in a variety of areas of time series processing, such as the anomaly detection in moving objects trajectories [2-3], the semantic analysis for the surgical sensor data streams [4], repeating pattern mining in audio streams [5-6], and human activity discovery [7-8]. Especially, it has been applied to the medical signals [9], like Electrocardiography (ECG) [10] and biological signals [11-13] for normal condition recognition and disease detection.

Discovering motifs for time series is an important and tough task. It has been proved that the subsequence clustering is meaningless in unsupervised data stream mining area, and the motif grouping in the discrete data stream mining has been applied as a replacement of the subsequence-clustering in the real-time series [14]. In this paper, we focus on two primary issues in the time series motif discovery: reducing the computational complexity and avoiding unexpected repetitions among different motifs and among instances of one motif. In an unsupervised context with little knowledge about the time series, it might be intractable to find all the motifs with different lengths by using exact and brute-force methods. There has been a series of work focusing on improving the time efficiency. One significant improvement is the method proposed by Minnen et al. [15], which has sub-quadratic time complexity in the time series length.

The subsequence trivial matching [16] and the overlapping among different motifs [1] are two types of motif repetition issues in the literature. To avoid trivial matching, some methods assumed that the instances of a motif do not overlap with each other at all [15]. We believe that, however, a more flexible and user-manageable mechanism is necessary to control the numbers and styles of the discovered patterns.

Enhancing the time and space complexity, and at the same time, guaranteeing an expected accuracy is always one of the top topics in data processing. Some motif discovery researchers used approximate solutions to get an acceptable computational complexity [17]. In this work, we propose an unsupervised Limited-length suffix array based Motif Discovery algorithm (LiSAM) for continuous time series, which is time and space efficient, and supports approximately discovering motifs in different lengths. We first convert the continuous time series to the discrete time series by using the Symbolic Aggregate approXimation procedure (SAX) [18], and then identify the different-length motifs based on the discrete time series. Our illustration of discrete motif discovery is on the basis of an exact substring matching procedure, however, we can easily embed the existing approximate substring matching methods, such as [19] and [20], in LiSAM. That is, we use the exact subsequence grouping of discrete time series to discover the approximate patterns of continuous time series. The distinctive contribution of LiSAM is as below:

(1) *LiSAM* can discover motifs in different lengths

*Corresponding Author: Le Sun; E-mail: sunle2009@gmail.com
DOI: 10.3966/160792642018111906020

(e.g., *maxLength* to *minLength* provided by users), avoid the unexpected trivial-matching by allowing user-defined overlapping degree (represented as α) between the instances of motifs, and support discovering motifs that overlap with each other in a specified degree (β). It can either be an automatic or semi-automatic algorithm by either manually setting all the parameters or by using default parameters (e.g., set $maxLength = 0.5 * |T|$ (T is a time series), $minLength = 2$, $\alpha = 0$ and $\beta = 0$).

(2) *LiSAM* is both space and time efficient. It has linear space complexity $O(N)$. We use a limited-length enhanced suffix array with linear space consumption to improve the space efficiency. In addition, in an extreme case that S has maximum LCP intervals, $O(LiSAM) = O(N+n)$, while in the case an interval has maximum child intervals, $O(LiSAM) = O(N+n^2)$, where N is the length of the raw time series T, and n is the length of the discrete time series S. If $N>>n$, the performance can be improved dramatically.

(3) We conduct extensive experiments based on both synthetic time series datasets to evaluate the performance of *LiSAM*. Experimental results show the high accuracy of *LiSAM* and its applicability in the pattern recognition of data streams such as ECG.

2 Background Knowledge

We briefly introduce the frequently used symbols (Table 1) and the basic concept of the enhanced suffix array in this section. Readers can refer to [21] for more details.

Table 1. Symboles and definitions

Concepts	Definitions
T	a continuous time series
Σ	a finite ordered alphabet
Σ^*	strings over Σ
Σ^+	Σ^* without null
S	a discrete time series over Σ with length $\|S\| = n$
$\sim$	$\sim\in\Sigma$, $\sim>\sigma$, $\forall\,\sigma\in\Sigma$
$S[i, j]$	substring of S between positions i and j
$suftab[suf]$	suffix array table of S
$presuf\,[pre]$	the suffix index of the previous position of the current suffix in *suftab*
$nextsuf\,[next]$	the suffix index of the next position of the current suffix in *suftab*
$S_{suftab[i]}$	the i^{th} suffix of S, $i\in[0, n]$
$lcptab[i]$	Longest common prefix (*LCP*) of $S_{suf[i-1]}$ and $S_{suf[i]}$
$bwttab[i]$ (bwt)	$S[suftab[i]-1]$, if $suf[i] > 0$; null, if $suf\,[i]=0$
l_ℓ-interval l_ℓ-[i, j]	an LCP interval from index i to index j With length ℓ
l-$[l, l]$	singleton interval (*SI*): $S_{suf[l]}$
NSI	non singleton interval
m_ℓ-$[i, j]$	m-interval: instances of l_ℓ interval

A suffix array of S is an integer array (*suftab*) having values $k\in[0, n]$. An enhanced suffix array (ESA) is a suffix array with a number of additional supporting arrays, where two of them (*lcptab* and *bwttab*) will be used in this paper. We use an example of S_{examp} = *aceaceacece* to describe the ESA that is shown in Table 2. The *suftab* keeps the starting positions of suffixes of S in ascending lexicographic order. The definition of *lcptab* is in Table 2. From Table 2, $lcptab[0] = 0$ and $lcptab[n] = 0$. To group the suffixes that have the longest common prefixes, the concept of LCP interval is proposed. We describe the definition of an LCP interval in Definition 1.

Table 2. An enhanced suffix array

index	*suf*	*lcptab*	*bwt*	$S_{suf[i]}$
0	0	0	*null*	aceaceacece~
1	3	6	*e*	aceacece~
2	6	3	*e*	acece~
3	1	0	*a*	ceaceacece~
4	4	5	*a*	ceacece~
5	7	2	*a*	cece~
6	9	2	*e*	ce~
7	2	0	*c*	eaceacece~
8	5	4	*c*	eacece~
9	8	0	*c*	cece~
10	10	0	*c*	e~
11	11	0	*e*	~

Definition 1. Given S and its enhanced suffix array, an interval $[i, j]$ of index (e.g., see Table 2), where $i, j\in[0, n]$ and $i < j$, is an LCP interval with LCP length ℓ if the following conditions are satisfied: (1) $lcptab[i] < \ell$; (2) $lcptab[k] \geq \ell$, $\forall\ k \in [i+1,\ j]$; (3) $lcptab[k] = \ell$, if $\exists\,k\in[i+1, j]$; (4) $lcptab[j+1] < \ell$. The LCP interval $[i, j]$ with LCP length ℓ can be represented as l_ℓ-$[i, j]$.

An LCP interval tree indicates the embedding and enclosing relations between LCP intervals. We describe an example of LCP tree of S_{examp} in Figure . We can see that the root of the LCP tree covers all the suffixes of S_{examp}. The child intervals are the intervals embedded in their father intervals. The leaf intervals do not enclose any *NSI*. A fast traversing procedure for LCP trees is defined in [22]. Note that in this paper we use l_ℓ to represent an l-interval with LCP length ℓ, while use m_ℓ to represent a motif interval (Def. 6}) with LCP length ℓ. In addition, we refer the normal *LCP intervals* to non-singleton intervals (*NSIs*).

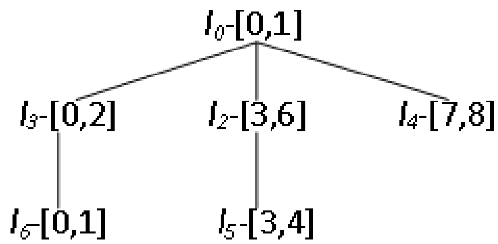


Figure 1. LCP tree of S_{examp}

3 Problem Formulation

A continuous time series T is a sequence of real values that have temporal properties. To identify the motifs of a time series, previous work has given different forms of motif definitions [23]. We summarize these definitions and present a comprehensive motif concept in Definition 2.

Definition 2. A motif M of a time series T is a set of similar subsequences $SQ = sq_0, ..., sq_{n-1}$ such that $n \geq 2$, and $\forall\, i, j \in [0, n\text{-}1]$, the length of $|sq_i| \geq 2$, $|sq_i \cap sq_j| \leq o$, and $Dis\ (sq_i,\ sq_j) \leq d$, where o is an overlapping threshold to constraint the overlapping length between two subsequences of M, Dis is a distance measure, and $d \geq 0$ is a small value to guarantee a certain similarity among subsequences. We call a subsequence of M as an *instance* of this motif.

Definition 3. Given two l-intervals l_{ℓ}-$[i_1, j_1]$ and l_{ℓ}-$[i_2, j_2]$, s_{k1} ($k1 \in [i_1, j_1]$) is an instance of $l_{\ell1}$, s_{k2} ($k2 \in [i_2, j_2]$) is an instance of $l_{\ell2}$, $sz_1 = |j_1\text{-}i_1\text{+}1|$: (1) instance s_{k1} is *α-covered* by s_{k2} if $\ell_1 < \ell_2$, s_{k1} overlaps with s_{k2} at sub-string s'' where $s'' \subseteq s_{k2}$ and $s'' \subseteq s_{k1}$, and $s'' > \alpha$, $s_{k1} \geq \alpha \geq (1/2)^* |s_{k1}|$. Or else, s_{k1} is *α-uncovered* by s_{k2}; *(2) interval* $l_{\ell1}$ is β-covered by $l_{\ell2}$, if h instances of $l_{\ell1}$ are covered by the instances of $l_{\ell2}$, where $sz_1\text{–}\beta < h \leq sz_1$, and h is a pre-defined threshold. Or else, $l_{\ell1}$ is *β-uncovered* (or uncovered) by $l_{\ell2}$.

A pattern of S is defined in Definition 4.

Definition 4. Given an alphabet set Σ and an approximate time series $S \in \Sigma^*$, a **pattern** of S is a time series pt that $1 \leq |pt| << |S|$, $pt \subseteq S$, and occurs k ($k \geq 2$) times in S at positions $\{p_1, ..., p_k\}$, $p_1 \neq ... \neq p_k$, where a position is the start point of an occurrence of pt in S.

In the above definition, we define that a pattern should occur at least twice in a time series. From the definition of l-interval, an l_{ℓ}-interval is composed of at least two suffixes that have the LCP of length ℓ. Therefore, an l-interval can be seen as a pattern of S, and the LCPs of the l-interval correspond to the occurrences of the pattern. However, the requirement on the minimum occurrence times of a pattern varies in different situations. For example, in a very long S (e.g., ≥10 thousands), the element that repeats a small number of times (e.g., <10 times) is meaningless for the time series analysis. Therefore, we define a general concept of an approximate motif of discrete time series in Definition 5.

Definition 5. Assume $u = S\,[a, b]$ ($a \leq b$) is an instance of an l-interval l_{ℓ}-$[i, j]$ of S. Given a lower bound $minT$ ($minT \geq 2$) of the pattern occurrences, if $\varepsilon = j\text{-}i\text{+}1 \geq minT$, and l_{ℓ} is uncovered by any other l-intervals of S, it is an **approximate motif** of S, represented as $mf = (\ell;\ P = p_1, \ldots, p_{\varepsilon}\,\})$, where $\ell = b - a + l$ ($l \geq 1$) is the *length* of mf, p_i is the start index of the occurrences of u in S, and ε is the **size** of the motif mf.

In the following description, a motif of S refers to an approximate motif. The relation between an l-interval and a motif of S is defined as an m-interval.

Definition 6. For an l-interval l_{ℓ}-$[i_1, j_1]$ of S, if the instances of l_{ℓ} is one-to-one matched to the occurrences of a motif mf = (ℓ; $\{suftab[i], \ldots, suftab[j]\}$)\$, then l_{ℓ} is an m-interval, represented as m_{ℓ}-$[i, j]$.

Based on Definition 6, motifs and m-intervals have the following relation.

Lemma 1. A motif of S corresponds to and only corresponds to one m-interval of S.

Proof. Given a motif $mf_u = (\ell;\ P_u = \{p_1,\ldots, p_{\varepsilon}\,\})$ of S, as $\varepsilon \geq 2$, then the subsequence u occurs at least twice in S. Based on the definition of LCP intervals and suffix array, the suffixes sf=$\{S[p_1, \sim],..., S[p_{\varepsilon}\,, \sim]\}$ are in one LCP interval l_{ℓ}-$[i, j]$, where $p_1,..., p_{\varepsilon} \in [i, j]$, ℓ=$|u|$ and ~ represents the end of S. Assume (1) $\exists\, k$, $k \in [i, j]$ that $s_1 = S[suftab[k], suftab[k\text{+}\ell\text{-}1]] = u$, but s_1 is not an occurrence of mf_u, i.e., $k \notin P_u$, which is contrast to the given condition that mf_u is a motif of S, because a motif needs to contain all the subsequences fitting one pattern. Assume (2) $\exists\, p_x \in P_u$ but $p_x \notin [i, j]$, and $\exists\, p_y \in P_u$ and $p_y \in [i, j]$, then ($s_1 = S[suftab[p_x], suftab[p_x\text{+}\ell\text{-}1]]$) $= u =$ ($s_2 = S[suftab[p_y], suftab[p_y\text{+}\ell\text{-}1]]$), that is, s_1 and s_2 are similar LCP and need to be in one LCP interval (suppose in l_{ℓ}' - $[i', j']$). As l_{ℓ} and l_{ℓ}' have one LCP u, they are the same l-interval, which is contrast to assumption (2). Lemma 1 is proved.

In the following sections, we refer an m-interval to a motif.

4 Limited-length Suffix-Array-Based Motif Discovery

The Limited-length Suffix-Array-Based Motif Discovery (LiSAM) Framework identifies motifs of S by determining the α-covering and β-covering degrees between instances of one l-interval and between different l-intervals, which is based on a bottom-up traversing process of identifying LCP intervals of the enhanced suffix array. The LiSAM is composed of two main algorithms: (1) βUncover (Alg. 1) determines whether or not an LCP interval is β-covered by other LCP intervals given a constraint $minT$ on the β-covering degree of a motif. From definition, the determination of β-covering is based on the α-covering degree. To identify the α-covering relations between instances, part (2) αUncovered (Alg. 4) is described, which determines the nontrivial matching instances of an LCP interval given a constraint on the α-covering degree between motifs. If an l-interval is β-uncovered, the instances of this interval form a motif.

4.1 Identify *β*-uncovered *l*-intervals for Discrete Time Series

In ESA, identifying LCP intervals is a bottom-up traversing process. When an LCP interval is being processed, its child intervals have been identified, so the child intervals can support the determination of β-covering of the LCP interval. We distinguish the case of an LCP interval having a single character (the singleChar interval) with the case that the interval is comprised of more than one character (the multiChar interval). We give Lemma 2 to identify the β-uncovered multiChar intervals.

Lemma 2. Given a multiChar LCP interval l_ℓ-$[i, j]$, its child intervals Θ, and the lower bound of the occurrence times of motifs $minT \geq 2$, let $\lambda = j - i + 1$, l_ℓ is β-uncovered by other l-intervals if any of the following conditions is satisfied:

(1) $|\Theta| = 0$, $\lambda = minT$ and $bwttab[i, j]$ are pair-wise different, i.e., $bwttab[i] \neq ... \neq bwttab[j]$;

(2) $|\Theta| = 0$, and $\exists \sigma_1 \neq ... \neq \sigma_\gamma$, $\sigma_{1, ..., \gamma} \in$ bwttab[i...j], $\mathrm{minT} + 1 \leq \gamma \leq \lambda$;

(3) $|\Theta| > 0$, $\exists\ l_{\ell 1}$ - $[w_1, z_1]$, $l_{\ell 1} \in \Theta$ and $\lambda_\theta = z_1 - w_1 + 1 \geq minT$, and $\exists r_1...r_k \in [w_1, z_1]$ and $h_1...h_k \in [i, j]$ but $\notin [w_1, z_1]$ that $bwttab[r_1] \neq bwttab[h_1]$, ..., $bwttab[r_k] \neq bwttab[h_k]$, $k \geq minT$.

(4) $|\Theta| > 1$, $\exists\ m_{\ell 1}$ - $[w_1, z_1]$, ..., $m_{\ell k}$ - $[w_k, z_k] \in \Theta$, $k \geq minT$, and $m_{\ell 1}$, ..., $m_{\ell k}$ are β-uncovered.

Proof:

(1) $|\Theta| = 0$, so the characters after the LCP subsequences of l_ℓ are pair-wise different, i.e., $S[suftab[i] + \ell] \neq S[suftab[j] + \ell]$. Meanwhile, $\lambda = minT$ and $bwttab[i] \neq ... \neq bwttab[j]$. Therefore, the instances of l_ℓ are not covered by any longer repeated sequences in S. Hence, l_ℓ is β-uncovered.

(2) if $\gamma > minT$, then at least $minT + 1$ characters in $bwttab[i, j]$ are different (assume $bwttab[k_1] \neq bwttab[k_2]$); and as $\Theta = 0$, the k_1 *th* and k_2 *th* LCP subsequences are not covered by any longer subsequences of its child intervals. So l_ℓ is β-uncovered.

(3) assume l_ℓ have one child interval c_θ, where $\lambda_\theta \geq minT$, $i \leq w_\theta \leq z_\theta \leq j$ and $\lambda > minT$. (a) Assume $\lambda - \lambda_\theta = 0$, then $l_\ell = c_\theta$, c_θ is not a child interval of l_ℓ. Assumption (a) is not true. (b) Assume $\lambda - \lambda_\theta < minT$, then there are $\lambda - minT$ instances of l_ℓ covered by the instances of c_θ, so interval l_ℓ is covered by interval c_θ, and l_ℓ is not a motif. Assumption (b) is not true. (c) as $\lambda - \lambda_\theta \geq minT$, then there are at least $minT$ instances of l_ℓ that are not covered by the instances of c_θ. In addition, $\exists \sigma_1 \neq ... \neq \sigma_\gamma$, $\sigma_{1,...,\gamma} \in bwttab[i...j]$, $minT < \gamma \leq \lambda$, based on the proof of (3), l_ℓ is β-uncovered.

(4) if $k = minT$, as $m_{\ell 1}$, ..., $m_{\ell k}$ are k motifs, the subsequences in all of the $minT$ intervals are pairwise different, so the interval l_ℓ, where $\ell < \ell_1, ..., \ell_{minT}$, cannot be covered by any of $\{m_{\ell 1}$ (as $\forall\ |m_{\ell t}| \geq minT$, $t \in [1, k]$, $t \neq 1$), ..., $m\ell_{minT}\}$, that is, the interval l_ℓ cannot be individually covered by any of its k child motifs. So l_ℓ is β-uncovered.

For singleChar intervals, the problem of determining their motif property is to avoid finding a shorter singleChar motif covered by a longer singleChar motif. Lemma 3 shows how to determine if a singleChar interval is β-uncovered.

Lemma 3. Given a singleChar interval l_ℓ - $[i, j]$ that its LCP subsequence, i.e., $S[suftab[i], suftab[i] + \ell - 1]$, is only comprised of one character (assume σ),

(1) if l_ℓ does not have child intervals, i.e., $|\Theta| = 0$ and $\exists \sigma_1 \neq ... \neq \sigma_\gamma$, $\sigma_{1,...,\gamma} \in bwttab[i...j]$, $minT + 1 \leq \gamma \leq \lambda$, then l_ℓ is β-uncovered;

(2) if $|\Theta| > 0$ and θ - $[w, z] \in \Theta$, that $\exists \sigma_1 \neq ... \neq \sigma_\gamma \neq \sigma$ and $\sigma_{1,...,\gamma} \in bwttab[w...z]$, where $\gamma > 0$, and $\exists \sigma'_1 \neq ... \sigma'_\lambda \neq \sigma$ and $\sigma'_{1,...,\lambda} \in bwttab[w'...z']$, where $z' - w' + 1 \geq 2$, $\lambda > 0$, $[w'...z'] \in [i...j]$ and $[w'...z']$ is β-uncovered by $[w...z]$;

Proof:

(1) As l_ℓ does not have child intervals, l_ℓ cannot be covered by an interval comprising LCP subsequences of $u' = S[suftab[k_1], ..., suftab[k_1] + \ell' - 1]$, where $k_1 \in [i, j]$, $\ell' > \ell$. In addition, as $\exists \sigma_1 \neq ... \neq \sigma_\gamma$, $\sigma_{1,...,\gamma} \in bwttab[i...j]$, $minT + 1 \leq \gamma \leq \lambda$, l_ℓ cannot be covered by an interval comprising LCP subsequences of $u'' = S[suftab[k_2] - 1, ..., suftab[k_2] - 1 + \ell'' - 1]$, where $k_2 \in [i, j]$, $\ell'' > \ell$. So l_ℓ is a β-uncovered.

(2) Assume $u = S[suftab[i] ... suftab[j] + \theta - 1]$ is the prefix of l_ℓ, and $u' = S[suftab[w] ... suftab[w] + \theta - 1]$ is the prefix of l_θ, and assume $\exists \sigma_1 \in bwttab[w...z]$ and $\exists \sigma_2 \in bwttab[w', z']$ that $\sigma_1 \neq \sigma$ and $\sigma_2 \neq \sigma$, then (1) any child interval l_θ cannot cover l_ℓ, since $z' - w' + 1 \geq 2$; (2) we prove that under condition 2 in Lemma 3, if l_ℓ is a singleChar interval with LCPs like $u = x_1, ..., x_\ell$, then not $\exists\ l_\theta$ (the strings of its *singleChar* LCP $u' = x_1, ..., x_\theta$, $(\theta > \ell)$) that cover l_ℓ. Assume such a l_θ exists, then the strings of the LCP of l_θ include all the stings whose prefixes with length θ are u', i.e., $\exists k\ (= z - w + 1)$ subsequences $u \in S$, and there must be $\eta\ (= k * (\theta - 1))$ *bwttabs* that $bwttab[r_1] = ... = bwttab[r_\eta] = \sigma$, $\eta = z' - w' + 1$ and $k + \eta = j - i + 1$; which means there must not exist $\sigma'_{1,...,\lambda} \neq \sigma$, $\lambda > 0$ in $bwttab[w', z']$. This is contradicting with condition 2 of Lemma 3, so the second statement (2) is correct. Combining statements (1) and (2), the singleChar interval l_ℓ is β-uncovered given condition 2 of Lemma 3.

Based on Lemma 2 and 3, we design the procedure of determining an LCP interval being β-uncovered in Algorithm 1. The procedure *sinChar*() (line 3) determines whether l_ℓ is a singleChar interval: if it is singleChar, return the character, otherwise, return null. The singleChar status of l-intervals can be determined in the construction process of the suffix array. *countUniqChar*() (line 4, Alg.2) calculates the number of different characters in $bwttab[i, j]$. If l_ℓ does not have children (*cd*); it is a singleChar interval; and it has at least *mt* different characters (*uc*) other than the *sc* character, then l_ℓ is a motif (lines 7-8 in Alg.1, point 1 in Lemma 3). If l_ℓ is a multiChar interval (*sc == null*) with more than *mt* unique characters, it is a motif (lines 7-8, point 2 in Lemma 2). For a multiChar interval, if it at least has *mt* children that are motif intervals (l_ℓ*.mcd.sz*), then it is a motif (lines 10-12, point 4 in Lemma2). For each interval, we can use an integer variable to keep the number of motifs of its children, and this integer value can be determined during the suffix array building process. Lines 13-18 are based on point 3 in Lemma 2, where *cu* represents the current child interval; *fcd* is the first child interval; *nt* and *la* are respectively the next and last child intervals of *cu*; and *lb* and *rb* are the left and right boundaries of an *l*-interval. The procedure *difCharPair*() (Line 14, Alg.3) compares a child interval (*cu*) of l_ℓ with the other part of l_ℓ (l'), where l' includes the intervals not covered by any child intervals of l_ℓ, and also the other child intervals apart from *cu*. If there are at least two pair of different character pairs in *bwttab*[*i*, *j*], that is, $\exists\, i_1, i_2, i_3, i_4 \in [i, j]$ that $bwttab[i_1] \neq bwttab[i_2]$ and $bwttab[i_3] \neq bwttab[i_4]$, then $cp \leftarrow true$.

Algorithm 1. Identify *β*-uncovered *l*-intervals

```
1: procedure βUNCOVERED(l_ℓ - [i,j], mi, ma, bwt,
   mt)
2:   sz ← j − i + 1
3:   sc ← sinChar(l_ℓ)
4:   uc ← countUniqChar(l_ℓ, singleChar)
5:   if sz == mt & bwt[i,j] are pair-wise different then
6:     return true
7:   else if l_ℓ.cd == nul & (sc ≠ nul & uc ≥ mt||uc >
     mt) then
8:     return true
8:   else
10:     if l_ℓ.mcd.sz ≥ mt & sc == null then
11:       return true
12:    end if
13:    for all cu ∈ l_ℓ.cd do
14:      cp ← cu.difCharPair(cu, l)
15:      if cp || cu == fcd & I < l_ℓ.fcd.lb & cp ||
         l_ℓ.cu.rb + 1 < l_ℓ.nt.lb & cp || cu == la &
         l_ℓ.la.rb + 1 < l_ℓ.rb & cp then
16:        return true
17:      end if
18:    end for
19:  end if
20:  return false
21:end procedure
```

Algorithm 2 calculates the number of different characters in an *l*-interval given that the interval is singleChar or multiChar (based on the value of *sc*). In line 2, if *l* does not have children, traverse the index tab (e.g., *i* in Table 3) of LCP table from *w* to *z*, and count the index if *bwttab*[*x*] is different with the other characters in *bwttab*[*i*, *j*] and different with the *sc* character (see the procedure *addCnt(c,cx)* in lines 24-29). The *addCnt()* indicates that if 1 is a multiChar interval (*sc* = *null* and *c* ≠ *sc*), or if it is a singleChar interval (without considering the indexes with *bwttab*[*x*] = *sc*, i.e., *c* ≠ *sc*), and the current character has not happened in *l'* (!*l .has(c)*), then count once and record the character in *l'*. Lines 7-20 count the pair-wise different characters in each child of *l* and in the indexes uncovered by any of its child, where *la* is the child interval of *l'* before *cu*; *nt* is the child interval after *cu*; and *lb* and *rb* are left and right bounds of an interval. Lines 15 checks which character is in the current child interval *cu*. If *cu* is not a singleChar interval or the character *sc* is not counted (*c* ≠ *sc*, in line 15), and the character *c* occurs in *cu* but not in *l'*, then this character *c* is counted (line 16), and is marked as happened in the interval *l'* (line 17).

Algorithm 2. Count the number of different characters in an *l*-interval

```
1:procedure COUNTUNIQCHAR(l' - [w, z], sc)
2:  if l_ℓ'.children == null then
3:    for all x ∈ [w,z] do
4:      addCnt(bwt[x],x)
5:    end for
6:  else
7:    for all cu ∈ l_ℓ'.cd do
8:      for all c ∈ bwttab[la.rb + 1... l_ℓ'.rb] do
9:        addCnt(c,cx)
10:       end for
11:       for all c ∈ bwttab[cu.rb + 1...nt.lb − 1] do
12:         addCnt(c,cx)
13:       end for
14:       for all c ∈ Sigma do
15:         if c ≠ sc & ! l_ℓ'.has(c) & cu.has(c) then
16:           l_ℓ'.cnt + +
17:           l_ℓ'.has(bwt[cx]) = true
18:         end if
19:       end for
20:     end for
21:   end if
22:   return cnt
23:end procedure
24:procedure ADDCNT(c,cx)
25:   if c ≠ sc & ! l_ℓ'.has(c) then
26:     l_ℓ'.cnt + +
27:     l_ℓ'.has(bwt[cx]) = true
28:   end if
29:end procedure
```

Table 3. Pre and nextsuf

i	pre	next	suf	sel.
0	-1	0	3	aceace
1	6	3	4	aceace
2	7	6	5	ace~
3	0	1	6	ceacea
4	1	4	7	ceace~
5	2	7	8	ce~
6	3	2	1	eaceac
7	4	5	2	eace~
8	5	8	9	e~
9	8	9	10	~

Algorithm 3 calculates the different character pairs between one child interval and the other part of l_ℓ. The inputs are two intervals (child intervals or l_ℓ's sub-intervals that are not covered by any child intervals of

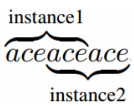
l_ℓ). l_ℓ is a motif if at least *mt* different character pairs (*cd*) are identified (line 8, point 4 in Lemma2); otherwise, count the next pair of characters in $l_{\ell 1}$ and $l_{\ell 2}$. In addition, if the current interval l_ℓ is a singleChar interval ($sc \neq nul$), and its two child intervals $l_{\ell 1}$ and $l_{\ell 2}$ both have at least 1 character that is different with the *sc* character, then l_ℓ is a motif (line 15-16, point 2 in Lemma3).

Algorithm 3. Count different character pairs between two intervals

```
1:procedure DIFCHARPAIR(lℓ1, lℓ2)
2:    cd = 0
3:    for all c1 ∈ Sigma do
4:        for all c2 ∈ Sigma & lℓ1.has(c1) & c ≠ sc do
5:            if c2 ≠ sc & lℓ2.has(c2) & c1 ≠ c2 then
6:                cd++
7:            end if
8:            if cd ≥ mt then
9:                return true
10:           else
11:               break the inner for-loop
12:           end if
13:       end for
14:   end for
15:   if sc ≠ nul & countUniqChar(lℓ1, sc) > 0 &
      countUniqChar(lℓ2, sc) > 0 then
16:       return true
17:   end if
18:   return false
19:end procedure
```

4.2 Identify α-uncovered Instances for Discrete Time Series

In section 3, we defined the concept of α-covering between instances of one interval. For example, in a time series s = *aceaceace*, if we expect a motif of length 6, we may get a motif with two instances:



where *instance*2 3-covers *instance*1. To control the α-covering degree, we introduce two tabs: *presuf* and *nextsuf* that respectively record the indexes of the previous suffix and the next suffix for the current suffix. An example of the two tabs is shown in Table 3.

The values of *pre* and *next* can be determined during the process of building suffix arrays, so it does not take extra time. The *pre* of the 0*th* suffix is −1 and the *next* of the last suffix is *length*(s).

Algorithm 4 shows how to identify the α-uncovered instances of an m-motif. In Algorithm 4, $\cap$ represents the overlapping part of two suffixes; $s[r..]$ represents the suffix starting from position r. If the index of the suffix (*ntS*) after the current suffix (*suf*) is in interval $[a,b]$, and the overlapping length between suffix $s[suftab[p]..]$ and suffix $s[suftab[ntS[p]]..]$ is less than the threshold value α, then the position $suf[p]$ is recorded as a start position of an αuncovered instance (lines 10-11). If the overlapping length between suffix $s[suf[p]..]$ and suffix $s[suf[ntS[p]]..]$ is over α, then continue checking the suffix after $ntS[p]$, until the checking step is over the *maD* (lines 8 to 15). As the LCP length of the current interval is ℓ, if an instance is *maD* far from the current instance, it is impossible that the two instances can α-cover each other. For each suffix, Algorithm 4 checks its α-covering instances by only iterating the suffixes from start positions afterwards. We temporally create an array ('*visited*' in lines 5, 6, 9) for the m-interval to record the visited status of each instance.

Algorithm 4. Identify αUncovered l-intervals

```
1:procedure αUNCOVERED(ϵ, mℓ - [a, b])
2:    maD = ℓ − α + 1
3:    for all p ∈ [a,b] do
4:        if mℓ.visited then
5:            P.add(p)
6:            mℓ.visited = true
7:        end if
8:        while q ≤ maD do
9:            if ntS[p] ∈ [a,b] & |s[suf[p]..] ∩
              s[suf[ntS[p]]..]| < α then
10:               P.add(p)
11:               mℓ[p].visited = true
12:           else if |s[suf[p]..] ∩ s[suf[ntS[p]]..]| ≥ α then
13:               mℓ[p].visited = true
14:           end if
15:       end while
16:   end for
17:   return P
18:end procedure
```

Algorithm 4 identifies α-uncovered instances given that the input interval is β-uncovered in terms of $\alpha = 1$. We can also interactively perform the algorithms βUncover and αUncover to determine the β-uncovered motifs in terms of different values of α by using the tab *pre*: in the process of βUncover, for each instance of an l-interval, we check both its pre- (i.e. suffixes with prior starting positions) and afterwards-suffixes simultaneously by using the chain-procedure of Algorithm 4 (for pre-suffixes, *next* can be simply replaced by *pre*). Specifically, we check each line in $[i, j]$ when $bwttab[i...j]$ is traversed in line 5 of Alg. 1, and determine whether this instance is overlapping with its previous instances *pre*. Remove it if it is overlapped with *pre*. In addition, in Alg. 2, we can check each position in $[i, j]$ in lines 3, 8 and 11, and remove this position if it is overlapped with its previous instance. At last, only the instances that are not overlapping with each other are used to decide if the current l-interval is

βUncovered.

5 Performance Evaluation and Complexity Analysis

In this section, we present the experimental results to show the efficiency of LiSAM. We insert patterns to random time series generated by gaussian white noise, and quantitatively measure the algorithm performance on the simulated data sets, in terms of the overlapping degree between the planted pattern and the discovered pattern of a time series (represented as *old*). In addition, time and space complexities of the proposed algorithm are analyzed. Our experiments are conducted on a windows 64-bit system with 3.2GHz CPU and 4 GB RAM, and is implemented by *Java*.

5.1 Accuracy and Inner Quality of Motifs

We extract patterns from six different ECG data streams [24], repeat each pattern 30 times and insert the repeated patterns to Gaussian white noise data streams separately. The information of the extracted patterns and the parameter settings is shown in the top part of Table 4. The first three datasets are from the UCR Time Series Classification Archive [24], and the other three are from the Physionet [25]. Particularly, the *nL* is the length of a piece of noise subsequence between two pieces of a pattern. We use the fixed-length intervals (i.e., length of noise subsequences) between two pattern subsequences to make the annotation of the pattern instances easy. Column *sL* sets the parameters of the SAX-based symbol conversion, representing the length of a subsequence that corresponds to a symbol. Columns *maxM* set the upper bounds of the lengths of the discovered patterns. The lower bounds of the lengths of the discovered patterns for all datasets are set as 10.

Table 4. Dataset settings & old and InDis performance

Datasets	nL	sL	maxM	old	inDis
ECG200	50	2	100	0.9892	0.0076
ECGfivedays	50	2	140	0.9924	0.0076
ECGtorse	100	10	1640	0.9947	0.0068
ECGtwa01	150	3	300	0.9933	0.0086
ECGsvdb800	150	2	170	0.9939	0.0112
ECGmitdb100	150	2	150	0.9966	0.006
LTDB14134	-	2	150	-	-
SVDB800	-	2	150	-	-
AHADB0001	-	2	120	-	-
CARTI01	-	2	100	-	-

We use *old* to measure the accuracy of the discovered motifs, which represents the overlapping degree between the inserted pattern (p_i) and the discovered pattern (d_j):

$$old = \frac{\sum i \sum j overlap(p_i, d_j)}{length(plantedPattern)} \tag{1.1}$$

The *old* values for each of the simulated ECG time series are shown in Table 4. We can see that the proposed motif discovery algorithm can identify the inserted patterns with very high accuracy (all over 0.9). We compare the shapes of the planted patterns and the discovered motifs in each of the six time series in Figure 2. In addition, we use the average pair-wise distances among instances (represented as *inDis*) of a motif to measure the dissimilarity degree of the instances of one discovered motif (e.g., motif m), which is calculated as:

$$inDis(m) = \frac{\sum i, j dis(m_i, m_j)}{m.len * m.size} \tag{1.2}$$

where m_i and m_j represent the *ith* and *jth* instances of *m*; and *m.len* is the length of this motif; *m.size* is the number of its instances, and *dis* is the Euclidean distance function. The average *inDis* value of each time series is shown in Table 4, and the distance distribution of each instance pair of the most frequent motif for each dataset is shown in Figure 3. We can see that the instances of one motif for each datasets are very close to each other, all of which have less than 0.1 average instance dis-similarities.

5.2 Pattern Discovery on Real Datasets

We use the proposed SAMOF algorithm to identify the most frequent patterns in four real ECG datasets: the MITBIH Long Term Database (LTDB), the Supraventricular Arrhythmia Database (SVDB), the American Heart Association Database (AHADB), and the St. Petersburg INCART Arrhythmia Database (CART) 0. Their information is listed in the bottom part of Table 4. For each dataset, we conduct pattern recognition in the first 30,000 samples (1:30000). We discover the most frequent motifs for each datasets, and present the motifs in Figure 4.

5.3 Time Complexity Analysis

The LiSAM mainly contains three steps: (1) discrete the time series based on SAX; (2) establish suffix array for the discrete time series and traverse the suffix array to find the LCP intervals; (3) determine the β-uncovered l-intervals.

If the length of a time series is N, the first step of time series discretion takes ON time. After discretion, if there are n symbols, the maximum time taken to build and traverse the suffix array (step 2) is $n + n = 2n$. The main part of Step 3 is the process of Algorithm 1.

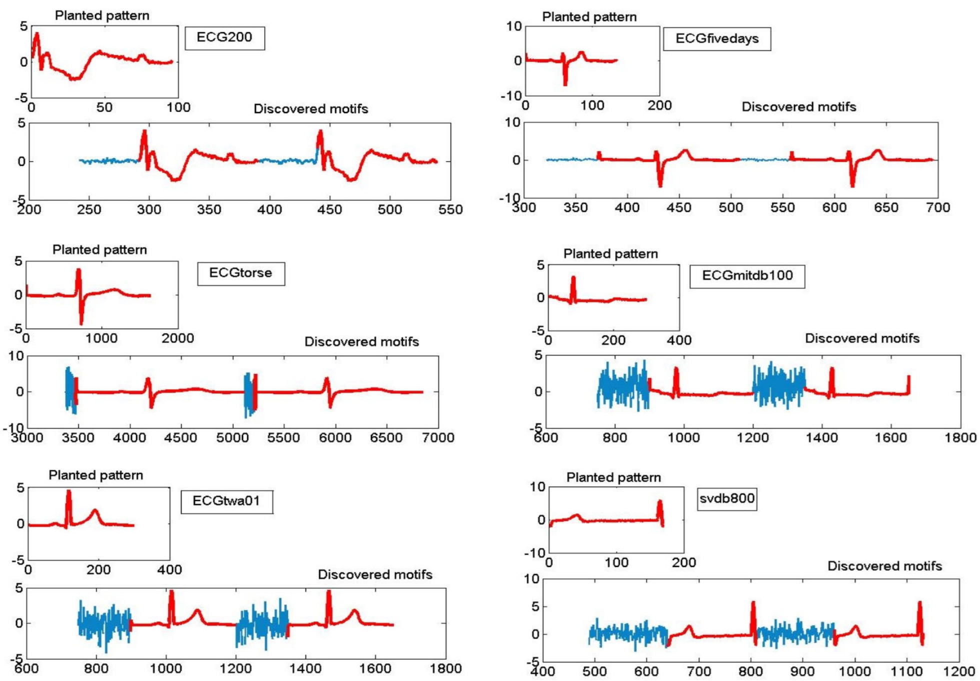


Figure 2. Planted patterns and discovered motifs

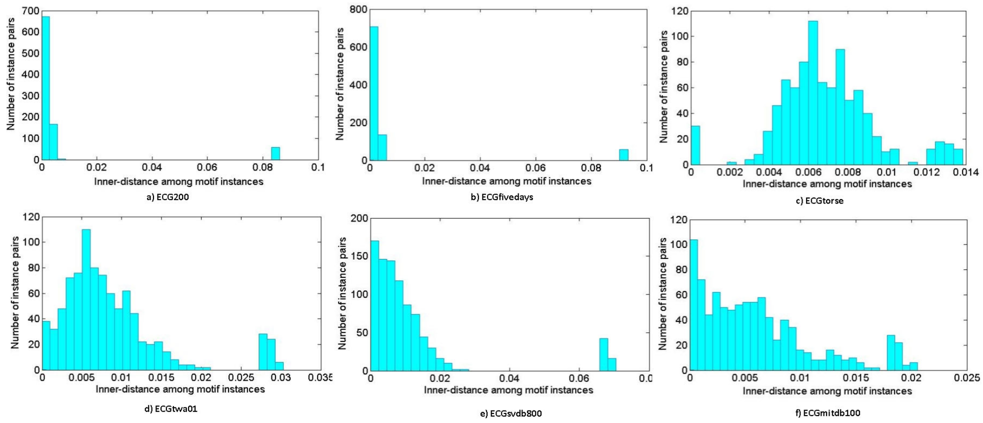


Figure 3. Distance distribution of instance pairs of the most frequent motif for six datasets

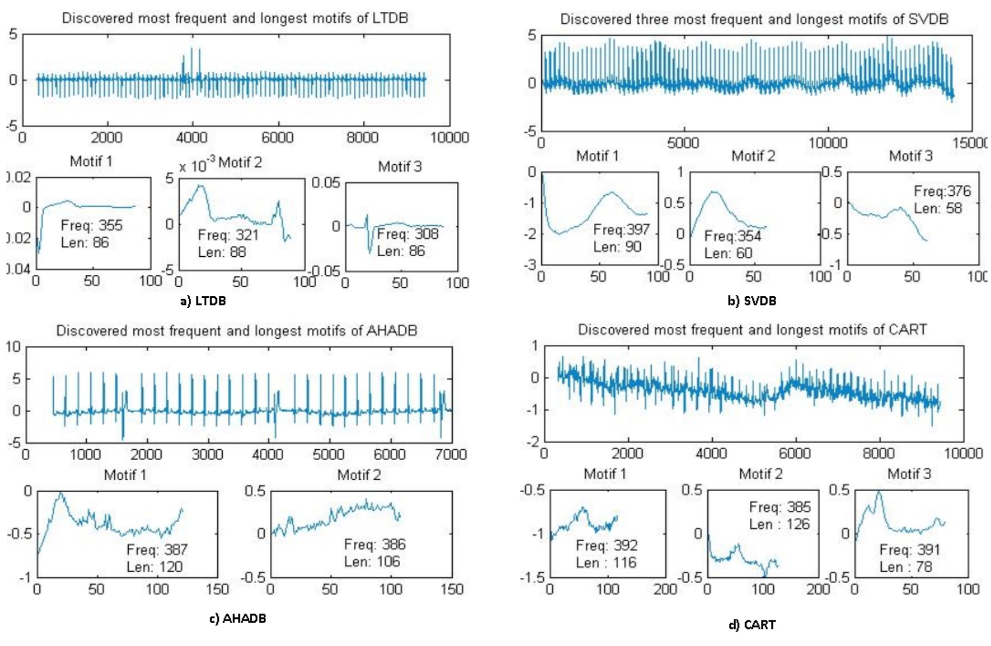


Figure 4. Discovered most frequent motifs of four real datasets

For an LCP interval l_ℓ − [i, j], the *hasSingleChar* function can be implemented during the suffix array construction process (line 3 in Alg.1). *countUniqChar*() function (line 7 in Alg.1, and Alg.2) takes maximum time $z \times r$, where $sz = |w - z|$ is the size of a child interval of l_ℓ, and r is the number of symbols in Σ. The three 'for-loop's in lines 6-8 in Alg.2 are actually a traverse of the index tab in [w, z]. As r is a constant normally less than 10, the O(*Alg*.2)) = O(sz). The time complexity of function *compInterv*() depends on the interval size sz and the complexity of *countUniqChar*(), so it is O(sz). Then the worst time complexity of LiSAM is: $O(LiSAM) = O(N + m_0 \times sz_0 + m_1 \times sz_1 + \cdots + m_K \times sz_K)$.

We may intuitively believe that the worst time complexity of LiSAM is $O(N{+}n^3)$. However, the values of K, m, and sz are interrelated with each other to influence the O(*LiSAM*). Lemma 4 gives their relations. We always exclude singleton intervals SI whenever we mention the l-intervals and their child intervals.

Lemma 4. Given a discrete time series S with length n, and an LCP tree LT of S.

(1) S has maximum $n - 1$ l-intervals, i.e., max(K) = $n - 1$, each LCP interval has at most 2 child non-singleton intervals (abbr. NSI), i.e., $m \le 2$, and the $max(sz) = n - 1$.

(2) S has minimum 1 l-interval (i.e., the root interval l[0…n-1]) that has 0 child NSI. Other than this case, the number of l-intervals of an LCP tree is a decreasing function of the child number of each LCP interval. That is, $K = f(1/c_k)$, where c_k is the child number of the kth interval.

(3) given an l_ℓ - [i, j], the number of its child intervals m is a decreasing function for the sizes of its child intervals: $m = f(1/sz)$.

Proof: We describe the problem of counting the LCP intervals as a problem of picking up elements from a set (see Figure 5). There are n sequential elements in S. Each time we remove any two adjacent elements (e.g., e_i, e_{i+1} in S_n) from S and combine these two elements as one new elements ($e_{i..i+1}$), and put this new elements back to S. For example, S_{n-1} in Figure 5 represents the S after the first time combination, and the number of elements in S_{n-1} is $n - 1$. We continue this process until there is only one element in S_1: $e_{1..n}$. In this process, we need to conduct the combination n−1 times in total. And each time we can combine both *SI*s and *NSI*s. We can see that each combination forms a new *NSI*, and this *NSI* has at most two *NSI* children. For the gth combination, there are n−g elements in the set S_{n-g}, g = 1,..., n−1. A child interval l 0-[w,z] of any l-intervals in LT has size $n - 1$ when it is after the (n − 2)th combination: $l_{\ell'}$ - [1, $n - 1$], which is the maximum size of a child interval.

We then prove that $n - 1$ is the maximum number of *NSI* in *LT*. If we remove k ($k > 2$) elements from S_{n-1}, …, S_{n-w}, where $w \ge 1$ and 1, …, w are not necessarily adjacent, and we remove 2 elements from

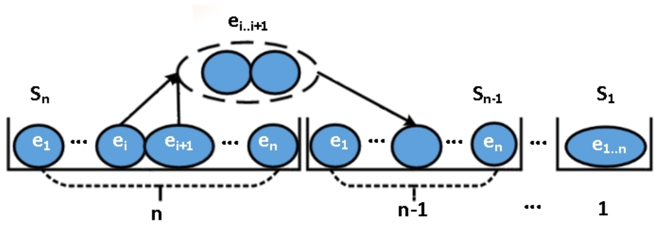


Figure 5. Number of LCP intervals

S_{n-v}, ∀ $v \in [1, n - 1]$, and $v \ne 1, \ldots, w$, then after w times combinations, it remains $n - w \times k$ elements in $S_{n-(w)}$, and requires $n - w \times k - 1$ times combination. So the overall combination times is $t = n{-}1{-}\,k(w - 1)$, as $k > 2$ and $w > 1$, so $t < n - 1 = max(K)$.

We call the behavior of combining more than one elements at one time as multi-combination (statement 1). This proof also indicates that as long as multi-combination happens (once or more than once and at any positions), the total number of LCP intervals will be decreased. Hence, the number of LCP interval is a decreasing function of the child number of each interval (statement 2).

Statement 3 is definite. When [i, j] is fixed, as the child intervals of l_ℓ cannot be overlap with each other, the increase of m will result in the decrease of z.

Based on Lemma 4, it is impossible that O(*LiSAM*) reaches $O(N + n^3)$. We consider two extreme cases:

- Assume S has maximum number of LCP intervals n−1, the root interval l^{n-1} has $sz_0 = n - 1$, and each interval (l^{n-1},⋯ ,l^1) has $max(m)$ = 2, then the time complexity of LiSAM is $O(LiSAM) = O(N + (n - 1) \times 2 + \cdots + 2 \times 2) = O(N + n)$;
- Assume C is a child interval of an l-interval in LT, has $sz = n - 2$, and has $m = floor((n\text{-}1)/2)$ NSI children, then C is the only child of the root interval [0, n−1], $K = 2 + m$, and each child of C has $sz = 2$ and has 0 children, then the time complexity is $O(N + 1 \times m \times (n - 1) + m \times 2) = O(N + n^2)$.

If S is highly compressed compared with T (i.e., $N \gg n$), the time complexity of LiSAM can be improved dramatically.

6 Conclusion and Future Work

In this paper, we proposed an algorithm LiSAM to resolve two important problems in discovering approximate time series motif: releasing the constraints of trivial matching between sub-sequences with different lengths and improving the time and space efficiency. We proposed two covering relations: α-covering between instances of l-intervals and β-covering between l-intervals to support the motif discovery. Experimental results showed the high accuracy of LiSAM on finding different-length motifs. In this paper, we focused on the exact discrete subsequence matching to identify clusters of sub-sequences with different lengths. In the future, we are going to explore the exact motif discovery based on the approximate motif grouping to further improve the motif identification accuracy and computational efficiency.

Acknowledgements

This paper is supported by the National Natural Science Foundation of China (Grants No 61702274) and the Natural Science Foundation of Jiangsu Province (Grants No BK20170958).

References

[1] J. Grabocka, N. Schilling, L. Schmidt-Thieme, Latent Time-Series Motifs, *Acm Transactions on Knowledge Discovery from Data*, Vol. 11, No. 1, pp. 1-20, August, 2016.

[2] J. Mao, T. Wang, C. Jin, A. Zhou, Feature Grouping-based Outlier Detection Upon Streaming Trajectories, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 29, No. 12, pp. 2696-2709, December, 2017.

[3] H. J. Jeong, M. J. Lee, C. E. Lee, S. H. Kim, Y. G. Ha, Machine Learning-Based Real-Time Anomaly Detection for Unmanned Aerial Vehicles with a Cloud Server, *Journal of Internet Technology*, Vol. 18, No. 4, pp. 823-832, July, 2017.

[4] J. Andreu-Perez, D. R. Leff, H. M. D. Ip, G. Z. Yang, From Wearable Sensors to Smart Implants-Toward Pervasive and Personalized Healthcare, *IEEE Transactions on Biomedical Engineering*, Vol. 62, No. 12, pp. 2750-2762, December, 2015.

[5] C. X. Mavromoustakis, G. Mastorakis, A. Bourdena, E. Pallis, Energy Efficient Resource Sharing Using a Traffic-oriented Routing Scheme for Cognitive Radio Networks, *IET Networks*, Vol. 3, No. 1, pp. 54-63, March, 2014.

[6] H. Xu, Z. Ou, Scalable Discovery of Audio Fingerprint Motifs in Broadcast Streams with Determinantal Point Process Based Motif Clustering, *IEEE/ACM Transactions on Audio Speech & Language Processing*, Vol. 24, No. 5, pp. 978-989, May, 2016.

[7] S. Wang, H. Yi, L. Wu, F. Zhou, N. N. Xiong, Mining Probabilistic Representative Gathering Patterns for Mobile Sensor Data, *Journal of Internet Technology*, Vol. 18, No. 2, pp. 321-332, March, 2017.

[8] P. Rashidi, D. J. Cook, Com: A Method for Mining and Monitoring Human Activity Patterns in Home-based Health Monitoring Systems, *ACM Transactions on Intelligent Systems and Technology*, Vol. 4, No. 4, pp. 1-20, September, 2013.

[9] J. Ma, L. Sun, H. Wang, Y. Zhang, U. Aickelin, Supervised Anomaly Detection in Uncertain Pseudo-periodic Data Streams, *ACM Transactions on Internet Technology*, Vol. 16, No. 1, Article No. 4, February, 2016.

[10] Y. J. Chang, W. T. Huang, A Novel Design of Data-driven Architecture for Remote Monitoring and Remote Control of Sensors over a Wireless Sensor Network and the Internet, *Journal of Internet Technology*, Vol. 12, No. 1, pp. 129-137, January, 2011.

[11] L. Deng, J. Zeng, X. Wang, An Improved Certificateless Encryption Scheme for Telecare Medicine Information Systems, *Journal of Internet Technology*, Vol. 18, No. 2, pp. 223-227, March, 2017.

[12] K. M. Al-Aubidy, A. M. Derbas, A. W. Al-Mutairi, Real-time Healthcare Monitoring System Using Wireless Sensor Network, *International Journal of Digital Signals and Smart Systems*, Vol. 1, No. 1, pp. 26-42, 2017.

[13] M. Dimkovski, A. An, A Bayesian Model for Canonical Circuits in the Neocortex for Parallelized and Incremental Learning of Symbol Representations, Neurocomputing, Vol. 149, No. 4, pp. 1270-1279, February, 2015.

[14] S. Aghabozorgi, A. S. Shirkhorshidi, T. Y. Wah, Time-series Clustering– A Decade Review, *Information Systems*, Vol. 53, No. 1, pp. 16-38, October-November, 2015.

[15] D. Minnen, C. L. Isbell, I. Essa, T. Starner, Discovering Multivariate Motifs using Subsequence Density Estimation and Greedy Mixture Learning, *Proceedings of the 22nd National Conference on Artificial Intelligence*, Menlo Park, Canada, 2007, pp. 615-620.

[16] R. Anirudh, P. Turaga, Geometry-based Symbolic Approximation for Fast Sequence Matching on Manifolds, *International Journal of Computer Vision*, Vol. 116, No. 2, pp. 161-173, January, 2016.

[17] T. Sun, H. Liu, H. Yu, C. P. Chen, Degree-pruning Dynamic Programming Approaches to Central Time Series Minimizing Dynamic Time Warping Distance, *IEEE Transactions on Cybernetics*, Vol. 47, No. 7, pp. 1719-1729, July, 2017.

[18] P. Nickerson, R. Baharloo, A. A. Wanigatunga, T. D. Manini, P. J. Tighe, P. Rashidi, Transition Icons for Time Series Visualization and Exploratory Analysis, *IEEE Journal of Biomedical and Health Informatics*, Vol. 22, No. 2, pp. 623-630, March, 2018.

[19] M. G. Baydogan, G. Runger, Time Series Representation and Similarity Based on Local Autopatterns, *Data Mining and Knowledge Discovery*, Vol. 30, No. 2, pp. 476-509, March, 2016.

[20] A. Bottrighi, G. Leonardi, S. Montani, L. Portinale, P. Terenziani, A Time Series Retrieval Tool for Sub-series Matching, *Applied Intelligence*, Vol. 43, No. 1, pp. 132-149, July, 2015.

[21] T. D. Wu, Bitpacking Techniques for Indexing Genomes: II. Enhanced Suffix Arrays, *Algorithms for Molecular Biology*, Vol. 11, No. 1, p. 9, April, 2016.

[22] M. I. Abouelhoda, S. Kurtz, E. Ohlebusch, Replacing Suffix Trees with Enhanced Suffix Arrays, *Journal of Discrete Algorithms*, Vol. 2, No. 1, pp. 53-86, March, 2014.

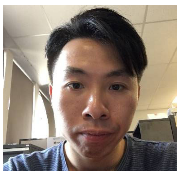
[23] A. Mueen, E. Keogh, Q. Zhu, S. S. Cash, M. B. Westover, N. Bigdely-Shamlo, A Disk-aware Algorithm for Time Series Motif Discovery, *Data Mining and Knowledge Discovery*, Vol. 22, No. 1, pp. 73-105, January, 2011.

[24] A. L. Goldberger, L. A. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, H. E. Stanley, Physiobank, Physiotoolkit, and Physionet: Components of a New Research Resource for

Complex Physiologic Signals, *Circulation*, Vol. 101, No. 23, pp. e215-e220, June, 2000.

Biographies

Le Sun is a Lecturer in School of Computer and Software, Nanjing University of Information Science and Technology, China. She got her Ph.D. (2016) from Center for Applied Informatics, Victoria University, Australia. Her research interests are: anomaly detection in data streams, cloud computing, and smart decision making.



Jinyuan He is a second-year Ph.D. student in Centre for Applied Informatics, Victoria University, Australia. He got his master degree (2015) in School of Software Engineering, Sun Yat-sen University, China. His research interests are: anomaly detection in data streams, cardiac disease detection in ECG, deep learning in bioinformatics.

Jiangang Ma received the Ph.D. degree from Victoria University. He is a research fellow at the Centre for Applied Informatics (CAI), Victoria University. His research interests include Data mining and Web services.

Hai Dong is a Lecturer at School of Science in RMIT University, Australia. He received a Ph.D. from Curtin University and a Bachelor degree from Northeastern University. He published over 70 research publications in international journals and conferences. His primary research interests include Services Computing, Cloud Computing, and IoT.

Yanchun Zhang is the Director of the Centre for Applied Informatics, Victoria University, Australia. He is an international research leader in databases, data mining, health informatics, web information systems, and web services. He has published over 220 research papers in international journals and conferences proceedings, and authored/edited 12 books.