Monitoring Range Motif on Streaming Time-Series

Shinya Kato(✉), Daichi Amagata, Shunya Nishio, and Takahiro Hara

Department of Multimedia Engineering Graduate School of Information Science and Technology, Osaka University, Yamadaoka 1-5, Suita, Osaka, Japan
kato.shinya@ist.osaka-u.ac.jp

Abstract. Recent IoT-based applications generate time-series in a streaming fashion, and they often require techniques that enable environmental monitoring and event detection from generated time-series. Discovering a range motif, which is a subsequence that repetitively appears the most in a time-series, is a promising approach for satisfying such a requirement. This paper tackles the problem of monitoring a range motif of a streaming time-series under a count-based sliding-window setting. Whenever a window slides, a new subsequence is generated and the oldest subsequence is removed. A straightforward solution for monitoring a range motif is to scan all subsequences in the window while computing their occurring counts measured by a similarity function. Because the main bottleneck is similarity computation, this solution is not efficient. We therefore propose an efficient algorithm, namely SRMM. SRMM is simple and its time complexity basically depends only on the occurring counts of the removed and generated subsequences. Our experiments using four real datasets demonstrate that SRMM scales well and shows better performance than a baseline.

Keywords: Streaming time-series · Motif monitoring

1 Introduction

Motif discovery is one of the most important tools for analyzing time-series [20]. Given a time-series t, its range motif is a subsequence that appears the most in t, i.e., a range motif is a frequently occurring subsequence [6,17]. As an example, in Fig. 1, we illustrate subsequences (red ones) which are repetitively appear in a streaming time-series of greenhouse gas emission [12], and the left most red subsequence is the current range motif. (We measure the similarity between subsequences by z-normalized Euclidean distance, thus the value scale in this figure is not a problem.) In this paper, we address the problem of monitoring a range motif (motif in short) of a streaming time-series, because recent IoT-based applications generate time-series in a streaming fashion [13].

Application Examples. It is not hard to see that this problem has a wide range of applications. For example, assume that a sensor device measures a sensor value and sends it to a server periodically, which constitutes a streaming

S. Hartmann et al. (Eds.): DEXA 2018, LNCS 11029, pp. 251–266, 2018.
https://doi.org/10.1007/978-3-319-98809-2_16

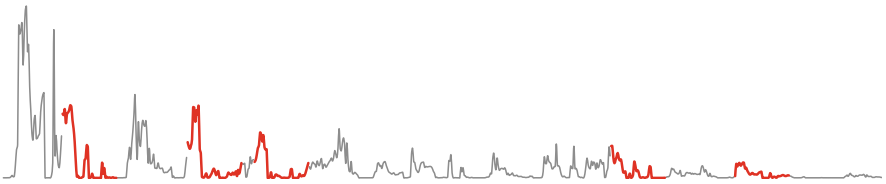


Fig. 1. An example of subsequences (red ones) which are repetitively appear and discovered in a streaming time-series of greenhouse gas emission [12]. We measure the similarity between subsequences by z-normalized Euclidean distance (that corresponds to Pearson correlation), and the current motif is the left most red subsequence. (Color figure online)

time-series. Assume further that a domain expert monitors the time-series, and if its motif changes as time passes, he/she can analyze some underlying phenomenon and form a hypothesis, e.g., sensor values have correlation with not only environmental but also temporal factors. Another example is event detection. Consider that we monitor the current motif and store it every minute. If the current motif is very different from the one obtained at the same time yesterday or we have a significant difference between the current and the previous motifs, it can be expected that there is an anomaly event.

Technical Overview. The above applications require monitoring the current motif in real-time while considering only recent data. We therefore employ a count-based sliding window setting, which considers only the most recent w data, and propose an efficient algorithm, namely SRMM (Streaming Range Motif Monitoring). When a given window slides, a new data is inserted into the window and the oldest data is removed from the window. That is, a new subsequence s_n, which contains the new data, is generated and the oldest one s_e, which contains the oldest data, is removed. A simple approach for updating the current motif, which is used as a baseline algorithm in this paper, is to scan all subsequences while comparing them with s_n and s_e. This can obtain the exact frequency count (the number of other subsequences that are similar to s_n and/or s_e) but incurs an expensive computational cost. SRMM avoids unnecessary computation by focusing on subsequences that can be the motif. The main idea employed in SRMM is to leverage PAA (Piecewise Aggregate Approximation) [7] and kd-tree [2]. This idea brings a technique which upper-bounds the frequency count of s_n with a light-weight cost, and enables pruning the exact frequency count computation. Even if we cannot prune the computation, we do not need to scan all subsequences. Actually, the upper-bounding collects a candidate of subsequences that may be similar to s_n. SRMM therefore needs to compare s_n only with the candidate subsequences.

Contributions. We summarize our contributions below.

- We address, for the first time, the problem of range motif (a subsequence that repetitively appears the most) monitoring on a streaming time-series under a count-based sliding window setting.
- We propose SRMM to efficiently update the current motif when a given window slides. SRMM is simple and efficient, and its time complexity is basically $O(\log(w-l)+m_n+m_e)$, where l is a given subsequence size and m_n and m_e are the upper-bound frequency counts of new and removed subsequences, respectively.
- We conduct experiments using four real datasets, and the results demonstrate that SRMM scales well and the performance of SRMM is better than that of the baseline.

Organization. We provide a preliminary in Sect. 2 and review some related works in Sect. 3. We present SRMM in Sect. 4 and introduce our experimental results in Sect. 5. Finally, Sect. 6 concludes this paper.

2 Preliminary

2.1 Problem Definition

A streaming time-series t is an ordered set of real values, which is described as $t=(t[1],t[2],...)$, where $t[i]$ is a real value. Because we are interested in an underlying pattern in t, we below define subsequence of t.

Definition 1 (Subsequence). *Given t and a length l, a subsequence of t, which starts at p is $s_p=(t[p],t[p+1],...,t[p+l-1])$.*

For ease of presentation, let $s_p[x]$ be the x-th value in s_p. To observe how many similar subsequences s_p have in t (i.e., the occurring count of s_p), we use Pearson correlation, which is a basic function to measure the similarity between time-series [10,15].

Definition 2 (Pearson correlation). *Given two subsequences s_p and s_q with length l, their Pearson correlation $\rho(s_p,s_q)$ is*

$$\rho(s_p,s_q)=1-\frac{\|\hat{s}_p,\hat{s}_q\|^2}{2l}. \tag{1}$$

We have $\rho(s_p,s_q)\in[-1,1]$. Note that $\|\hat{s}_p,\hat{s}_q\|$ computes the Euclidean distance between $\hat{s}_p$ and $\hat{s}_q$, and

$$\hat{s}_p[i]=\frac{s_p[i]-\mu(s_p)}{\sigma(s_p)},$$

where $\mu(s_p)$ and $\sigma(s_p)$ are the average and the variation of $(s_p[1],s_p[2],...,s_p[l])$, respectively. Now we see that $\hat{s}_p$ is the z-normalized version of s_p, and Pearson

correlation can be converted to the z-normalized Euclidean distance $d(\cdot,\cdot) = \|\cdot,\cdot\|$, i.e., from Eq. (1),

$$d(\hat{s}_p, \hat{s}_q) = \sqrt{2l(1-\rho(s_p, s_q))}. \tag{2}$$

It is trivial that the time complexity of computing Pearson correlation is $O(l)$. We next define subsequences which are similar to s_p.

Definition 3 (Similar subsequence). *Given s_p, s_q, and a threshold θ, we say that s_q (s_p) is similar to s_p (s_q) if*

$$\rho(s_p, s_q) \geq \theta \Leftrightarrow d(\hat{s}_p, \hat{s}_q) \leq \sqrt{2l(1-\theta)}. \tag{3}$$

It can be easily seen that s_p and s_{p+1} can be similar to each other, but such a pair is not interesting to obtain a meaningful result. Such overlapping subsequences are denoted by trivial matched subsequences [5,17].

Definition 4 (Trivial match). *Given s_p, its trivial matched subsequences s_q satisfy that $p - l + 1 \leq q \leq p + l - 1$. $\overline{S_p}$ denotes the set of trivial matched subsequences of s_p.*

Now we consider the occurring count of s_p, $score(s_p)$ in other words.

Definition 5 (Score). *Given t, l, and θ, the score of a subsequence $s_p \in t$ is defined as:*

$$score(s_p) = |\{s_q \,|\, s_q \in t, \rho(s_p, s_q) \geq \theta, s_q \notin \overline{S_p}\}|. \tag{4}$$

Here, many applications including the ones in Sect. 1 care only recent data [8,14]. Hence, as with existing works that study streaming time-series [4,9], we employ a count-based sliding window setting, which monitors only the most recent w values. That is, a streaming time-series t in the window is represented as $t = (t[i], t[i+1], ..., t[i+w-1])$ where $t[i+w-1])$ is the newest value, and there are $(w - l + 1)$ subsequences in the window when l is given. When the window slides, we have a new subsequence which consists of the most recent l values. At the same time, the oldest value is removed from the window, so the oldest subsequence expires. We would like to monitor the subsequence of t with the maximum score in this setting. Let S be the set of all subsequences in a given widow with size w, and formally, our problem is:

Definition 6 (Range motif monitoring problem). *Given t, l, θ, and w, the problem in this paper is to monitor the current range motif s^* that satisfies*

$$s^* = \arg\max_{s \in S} score(s).$$

If the context is clear, range motif is called motif simply.

2.2 Baseline Algorithm

Because this is the first work that tackles this problem, we first provide a naive solution that can monitor the exact result. Section 1 has already introduced the solution, which updates the scores of all subsequences in the window by comparing them with the expired and new subsequences, whenever the window slides. As mentioned earlier, there are $(w - l + 1)$ subsequences in the window and each score computation requires $O(l)$ time. Therefore, the time complexity of this solution is $O((w - l)l)$.

We can intuitively see that, for a subsequence, comparing it with all subsequences incurs redundant computation cost, because the subsequence is interested only in its similar subsequences. To remove such a redundant cost, we propose a technique that efficiently identifies subsequences whose scores need to be updated.

3 Related Work

We introduce existing works that tackle the problem of motif discovery. It is important to note that the term *motif* is sometimes used in different meaning, as claimed in [6]. The first definition of motif is the same as that in this paper. On the other hand, some works, e.g., [10,14,15], use motif as the closest subsequence pair in a time-series. In this section, if referred literatures study the problem of discovering the closest subsequence pair, we say that it is pair-motif discovery problem.

3.1 Pair-Motif Discovery Problem

This problem suffers from its quadratic time complexity w.r.t. the number of subsequences, thus it is not trivial to make exact algorithms scale well. Literature [15] first proposed an exact algorithm MK that exploits triangle inequality. MK selects some subsequences as reference points, and utilize them to obtain upper-bound distances when it compares a given subsequence and another one. However, its time complexity is still quadratic. To scale better, [10] proposed Quick-Motif algorithm. Quick-Motif builds an subsequence index in online to reduce the number of subsequence comparisons. Its experiments show that Quick-Motif significantly outperforms MK. Recently, an offline index approach, called Matrix Profile, was proposed in [21,22]. For all subsequences, this index maintains the distances to other subsequences with the largest similarity. This index makes an online pair-motif discovery algorithm fast [22].

The above studies consider static time-series. The first attempt to monitor the pair-motif is performed in [14]. For each subsequence, the algorithm proposed in [14] maintains its nearest neighbor and reverse nearest neighbor subsequences to deal with the pair-motif update. Literature [8] has optimized a data structure for pair-motif monitoring and the algorithm proposed in [8] outperforms the algorithm of [14].

3.2 Range-Motif Discovery Problem

Patel et al. proposed an approximate algorithm to discover a range motif efficiently [17]. In this algorithm, each subsequence is converted to a string sequence by SAX [11]. Similar to this algorithm, Castro and Azevedo proposed a range motif discovering algorithm [3] that employs iSAX [19]. Both SAX and iSAX approximate a given time-series, thus the discovered motif is not guaranteed to be exact. Some probabilistic algorithms are proposed in [5,20], and again, this approach does not guarantee the correctness. Literature [6] proposed a learning-based motif discovery algorithm. This algorithm requires pre-processing step, thus is hard to be applied in streaming setting. The above literatures consider only a static time-series.

Although [1] considers a streaming time-series, it aims to discover a rare subsequence that has some similar subsequences but with some *very low probability*. The algorithm proposed in [1] also employs approximate approaches (SAX and Bloom filter). [16] also considers a streaming time-series, but this literature considers a distance between subsequences under SAX representation. As can be seen above, the existing works basically consider approximate solutions. In this paper, we provide an *exact* solution for efficient motif monitoring.

4 SRMM: Streaming Range Motif Monitoring

We first note that *the score of each subsequence in the window increases at most one* when the window slides, which can be seen from Definition 5 and the property of count-based sliding window. This observation suggests that the current motif does not change frequently and the score of the new subsequence often does not reach $score(s^*)$.

Let s_n be the new subsequence, and if we can know that $score(s_n) < score(s^*)$ with a *light-weight cost*, we can efficiently monitor the exact motif. To achieve this, we propose a technique that obtains an upper-bound of $score(s_n)$ efficiently and prunes unnecessary exact score computation. We introduce this technique in Sect. 4.1. Recall that the oldest subsequence is removed from the window, which makes the scores of some subsequences decrease by one. This may affect s^*. SRMM can efficiently identify the subsequences whose scores may decrease, which is described in Sect. 4.2. Finally, We elaborate the overall algorithm of SRMM and provide its time complexity in Sect. 4.3.

4.1 Upper-Bounding

First, we obtain an upper-bound of Pearson correlation between s_n and $s \in S$, which corresponds to a lower-bound of the z-normalized distance (see Eq. (2)). We use PAA [7], a dimensionality reduction algorithm, to achieve this. Recall that a subsequence s_p is represented as $(s_p[1], s_p[2], ..., s_p[l])$. This implies that it can be regarded as a point on an l-dimensional space $\mathbb{R}^l$, i.e., a subsequence is an l-dimensional point.

Given a dimensionality $\phi < l$, PAA transforms an l-dimensional point into a ϕ-dimensional point. Let $\hat{s}_p^\phi$ be the transformed $\hat{s}_p$. Each value of $\hat{s}_p^\phi$ is described as

$$\hat{s}_p^\phi[i] = \frac{\phi}{l} \sum_{j=\frac{l}{\phi}i}^{\frac{l}{\phi}(i+1)-1} \hat{s}_p[j].$$

PAA has the following lemma.

Lemma 1 [7]. *Given two subsequences $\hat{s}_p$ and $\hat{s}_q$, we have*

$$\sqrt{\frac{l}{\phi}} dist(\hat{s}_p^\phi, \hat{s}_q^\phi) \leq dist(\hat{s}_p, \hat{s}_q). \tag{5}$$

From PAA, we can obtain a lower-bound of the Euclidean distance between $\hat{s}_p$ and $\hat{s}_q$, i.e., an upper-bound of $\rho(s_p, s_q)$ in $O(\phi)$ time. If $\sqrt{\frac{l}{\phi}} dist(\hat{s}_p^\phi, \hat{s}_q^\phi) > \sqrt{2l(1-\theta)}$, s_q is not similar to s_p (see Definition 3), thus we can safely prune the exact distance computation between $\hat{s}_p$ and $\hat{s}_q$. Given $\hat{s}_n$, an upper-bound of $score(s_n)$ can be obtained if we compute $\sqrt{\frac{l}{\phi}} dist(\hat{s}_n^\phi, \hat{s}_p^\phi)$ for $\forall s_p \in S \backslash \overline{S}_n$. However, this approach is still expensive, incurs $O(\phi(w-l))$ time, and s_n is interested only in s_p such that $\sqrt{\frac{l}{\phi}} dist(\hat{s}_n^\phi, \hat{s}_p^\phi) \leq \sqrt{2l(1-\theta)}$. To obtain such s_p efficiently, we employ a kd-tree [2], which is a binary tree for an arbitrary dimensional space. The behind idea of employing a kd-tree is that kd-tree supports efficient data insertion, deletion, and range query processing.

Assume that all transformed subsequences in the window are indexed by a kd-tree. Now we see that s_p, such that $\sqrt{\frac{l}{\phi}} dist(\hat{s}_n^\phi, \hat{s}_p^\phi) \leq \sqrt{2l(1-\theta)}$, is obtained by a range query where the query point is $\hat{s}_n^\phi$ and the distance threshold is $\sqrt{2\phi(1-\theta)}$. Then we have the following theorem.

Theorem 1. *Assume that we have a new subsequence s_n, a distance threshold $\sqrt{2l(1-\theta)}$, and a kd-tree that maintains all subsequences, except the l most recent ones, which are transformed by PAA. A range query on the kd-tree, where its query point and a distance threshold respectively are $\hat{s}_n^\phi$ and $\sqrt{2\phi(1-\theta)}$, returns S_n^{in} which is a set of transformed subsequences $\hat{s}_p^\phi$ such that $dist(\hat{s}_n^\phi, \hat{s}_p^\phi) \leq \sqrt{2\phi(1-\theta)}$. Let $|S_n^{in}| = m_n$, and we have $m_n \geq score(s_n)$.*

Proof. We want s_p that satisfies $\sqrt{\frac{l}{\phi}} dist(\hat{s}_n^\phi, \hat{s}_p^\phi) \leq \sqrt{2l(1-\theta)}$, which can be seen from Lemma 1. This inequality derives $dist(\hat{s}_n^\phi, \hat{s}_p^\phi) \leq \sqrt{2\phi(1-\theta)}$. Next, the l most recent subsequences can be trivial matched subsequences of s_n, thereby they are not necessary to compute $score(s_n)$. Theorem 1 therefore holds. □

Example 1. Figure 2 illustrates a set of transformed subsequences where $\phi = 2$, i.e., they are 2-dimensional points. To obtain an upper-bound score of s_n, we

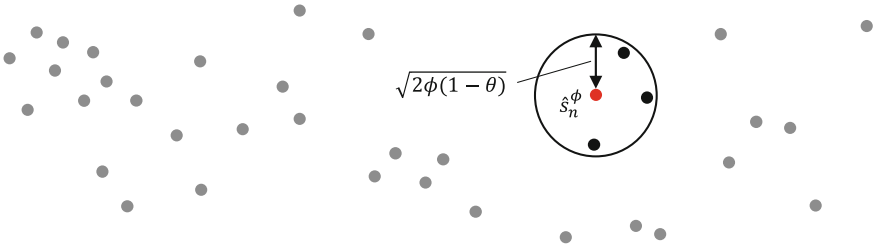


Fig. 2. An example of upper-bounding of $score(s_n)$, where $\phi = 2$. The red point is s_n and $m_n = 3$, since there are three points within the circle centered at $\hat{s}_n^{\phi}$ with the radius $\sqrt{2\phi(1-\theta)}$. (Color figure online)

set $\sqrt{2\phi(1-\theta)}$ as a distance threshold and execute a range query centered at $\hat{s}_n^{\phi}$ (the red point). As a query answer, we have three (black) points, which are efficiently retrieved by using a kd-tree, and we have $m_n = 3$.

Theorem 1 provides the following corollary.

Corollary 1. *If $score(s) \geq m_n$ where $s \in S \backslash \{s_n\}$, s_n cannot be the current motif, thus we can safely prune the exact computation of $score(s_n)$.*

Due to Theorem 1, we do not index the l most recent subsequences by a kd-tree. Here, the time complexity of a range query on a kd-tree is $O(\log n + m)$ where n and m are the cardinalities of data in the kd-tree and of data satisfying the distance threshold. The time complexity of the upper-bounding is hence $O(\log(w-l) + m_n)$, and we have $(\log(w-l) + m_n) \ll w$.

4.2 Identifying the Subsequences Whose Scores Can Decrease

When the window slides, the oldest subsequence expires, which makes the scores of some subsequences decrease. One may consider that a range query centered at the expired subsequence can solve this score updates. However, such a duplicate evaluation is not efficient. We overcome this problem by utilizing two lists for each subsequence s_p, similar list SL_p and possible similar list PL_p.

Definition 7 (Similar list). *The similar list of s_p, SL_p, is a set of tuples of subsequence identifier q and $\rho(s_p, s_q)$, i.e., $SL_p = \{\langle q, \rho(s_p, s_q)\rangle \,|\, s_q \in S \backslash S_p, \rho(s_p, s_q) \geq \theta\}$.*

Definition 8 (Possible similar list). *The possible similar list of s_p, PL_p, is a set of identifiers of subsequences s_q such that $dist(\hat{s}_p^{\phi}, \hat{s}_q^{\phi}) \leq \sqrt{2\phi(1-\theta)}$, $s_q \notin S_p$, and $\langle q, \cdot\rangle \notin SL_p$.*

In a nutshell, when we compute an upper-bound score of s_p by a range query, we add q, such that $dist(\hat{s}_p^{\phi}, \hat{s}_q^{\phi}) \leq \sqrt{2\phi(1-\theta)}$, into PL_p. We also add p into PL_q. In addition, when we compute $\rho(s_p, s_q)$, we remove q (p) from PL_p (PL_q), and if $\rho(s_p, s_q) \geq \theta$, we update SL_p and SL_q. Now we have two lemmas.

Algorithm 1. SRMM (expiration case)

Input: s_e: the expired subsequence
Output: s^*_{temp}: a temporal motif
1 Delete $\hat{s}^{\phi}_e$ from kd-tree, $f \leftarrow 0$
2 **for** $\forall p \in SL_e$ **do**
3 $SL_p \leftarrow SL_p \backslash \langle e, \cdot \rangle$
4 **if** $s_p = s^*$ **then**
5 $f \leftarrow 1$
6 **for** $\forall p \in PL_e$ **do**
7 $PL_p \leftarrow PL_p \backslash \{e\}$
8 **if** $s^* = s_e$ **then**
9 $f \leftarrow 1$, $s^* \leftarrow \varnothing$
10 $s^*_{temp} \leftarrow s^*$
11 **if** $f = 1$ **then**
12 **for** $\forall s_p \in S$ *such that* $|SL_p| + |PL_p| \geq score(s^*_{temp})$ **do**
13 $s^*_{temp} \leftarrow$ Motif-Update(s_p, s^*_{temp})

Lemma 2. $|SL_p| + |PL_p| \geq score(s_p)$.

Lemma 3. *The subsequences s_q, whose scores can decrease due to the expiration of s_e, satisfy that $q \in PL_e$ or $\langle q, \cdot \rangle \in SL_e$.*

Both Lemmas 2 and 3 can be proven by Definitions 7 and 8. Now we see from Lemma 3 that SL_q and PL_q can be updated in $O(1)$ time, so its total update time is $O(|SL_e| + |PL_e|)$.

4.3 Overall Algorithm

We present the detail of SRMM, which exploits the techniques introduced in Sects. 4.1 and 4.2. When the window slides, we first deal with the expired subsequence and obtains a temporal motif s^*_{temp}. After that, we verify whether the new subsequence can be s^*.

Dealing with Expired Subsequence s_e. Algorithm 1 details how SRMM deals with the expired subsequence. Given the expired subsequence s_e, SRMM deletes $\hat{s}^{\phi}_e$ from the kd-tree, which is done in $O(\log(w - l))$ time, and sets a flag $f = 0$ (line 1). Then, according to Lemma 3, SRMM deletes $\{e\}$ and $\langle e, \cdot \rangle$ from all PL_p and SL_p such that $p \in PL_e$ or $\langle p, \cdot \rangle \in SL_e$ (lines 2–9). Note that if $score(s^*)$ decreases or $s^* = s_e$, we set $f = 1$. Last, if $f = 1$, the current motif can be changed. From Lemma 2, we see the subsequences s_p which can be the motif have to satisfy $|SL_p| + |PL_p| \geq score(s^*_{temp})$. SRMM therefore computes the exact scores of such s_p and obtains a temporal motif s^*_{temp} (line 13), through Motif-Update(s_p, s^*_{temp}), which is introduced later.

We next confirm that the obtained temporal motif is really the current motif or the new subsequence can be the current motif.

Algorithm 2. SRMM (insertion case)

Input: s_n: the new subsequence, s^*_{temp}: a temporal motif
Output: s^*: the current motif

1 Compute $\hat{s}^{\phi}_n$ by PAA
2 Insert $\hat{s}^{\phi}_{n-l}$ to kd-tree
3 $SL_n \leftarrow \varnothing$
4 $PL_n \leftarrow$ Range-Search$(\hat{s}^{\phi}_n, \sqrt{2\phi(1-\theta)})$
5 **for** $\forall p \in PL_n$ **do**
6 **if** $s_p = s^*_{temp}$ **then**
7 Compute $\rho(s_p, s_n)$
8 **if** $\rho(s_p, s_n) \geq \theta$ **then**
9 $SL_p \leftarrow SL_p \cup \langle n, \rho(s_p, s_n)\rangle$, $SL_n \leftarrow SL_n \cup \langle p, \rho(s_p, s_n)\rangle$
10 $PL_n \leftarrow PL_n \backslash \{p\}$
11 **else**
12 $PL_p \leftarrow PL_p \cup \{n\}$
13 **if** $|SL_p| + |PL_p| \geq score(s^*_{temp})$ **then**
14 $s^*_{temp} \leftarrow$ Motif-Update(s_p, s^*_{temp})
15 **if** $|SL_n| + |PL_n| \geq score(s^*_{temp})$ **then**
16 $s^* \leftarrow$ Motif-Update(s_p, s^*_{temp})
17 **else**
18 $s^* = s^*_{temp}$

Dealing with New Subsequence s_n. Algorithm 2 illustrates how SRMM updates the current motif. SRMM first obtains $\hat{s}^{\phi}_n$ by PAA and inserts $\hat{s}^{\phi}_{n-l}$ into the kd-tree (lines 1–2). Note that s_{n-l} is the most recent subsequence that does not overlap with s_n. (Recall that our kd-tree does not maintain the l most recent transformed subsequences.) Then SRMM sets $SL_n = \varnothing$ and obtains PL_n by a range query, as explained in Sect. 4.1 (lines 3–4). For $\forall p \in PL_n$, PL_p also needs to be updated. If $s_p = s^*_{temp}$, SRMM computes $\rho(s_p, s_n)$ to obtain $score(s_p)$, and then updates SL_p, SL_n, and PL_n (lines 6–10). On the other hand, if $s_p \neq s^*_{temp}$, PL_p is updated and SRMM checks whether $|SL_p| + |PL_p| \geq score(s^*_{temp})$ or not. In the case where it is true, SRMM executes Motif-Update(s_p, s^*_{temp}) and updates s^*_{temp} if necessary (line 14). Last, if $|SL_n| + |PL_n| \geq score(s^*_{temp})$, SRMM executes Motif-Update(s_n, s^*_{temp}) to verify the current motif (line 15–16). Otherwise, we can guarantee that s^*_{temp} is now s^* (line 18).

Speeding Up Verification. In Motif-Update(s_n, s^*_{temp}), we confirm whether or not $\rho(s_n, s^*_{temp}) \geq \theta$, update their similar and possible similar lists, and replace s^*_{temp} if necessary. We see that updating similar and possible similar lists requires $O(1)$ time, so if we can relieve the confirmation cost, the motif verification cost is reduced. We achieve this by using the following theorem.

Theorem 2. *When s_n, s_p where $p \in PL_n$, s_q where $q \in PL_n \wedge \langle q, \rho(s_p, s_q)\rangle \in SL_p$, and θ are given, we have $\rho(s_n, s_q) \geq \theta$ if $dist(\hat{s}_n, \hat{s}_p) + dist(\hat{s}_p, \hat{s}_q) \leq \sqrt{2l(1-\theta)}$.*

Proof. Recall that $dist(\cdot, \cdot)$ is the z-normalized Euclidean distance. Therefore, from triangle inequality and Eq. (3), Theorem 2 holds. □

Recall that if $|SL_n| + |PL_n| \geq score(s^*_{temp})$, we need to compute $score(s_n)$. We accelerate this verification, i.e., Motif-Update(s_n, s^*_{temp}) by exploiting Theorem 2. As a reference subsequence, we utilize s_p which is the nearest neighbor to s_n, in the ϕ-dimensional space, among a set of subsequences $s_{p'}$ such that $p' \in PL_n$ and $SL_{p'} \neq \varnothing$. Note that s_p is obtained during Range-Search$(\hat{s}^\phi_n, \sqrt{2\phi(1-\theta)})$. First, we compute $dist(\hat{s}_n, \hat{s}_p)$. Then, for $\forall q \in PL_n$, we compute $dist(\hat{s}_n, \hat{s}_p) + dist(\hat{s}_p, \hat{s}_q)$ if $\langle q, \cdot\rangle \in SL_p$. If we have $dist(\hat{s}_n, \hat{s}_p) + dist(\hat{s}_p, \hat{s}_q) \leq \sqrt{2l(1-\theta)}$, we do not need to compute $dist(\hat{s}_n, \hat{s}_q)$. Therefore, we compute $dist(\hat{s}_n, \hat{s}_q)$ only in cases where we have $dist(\hat{s}_n, \hat{s}_p) + dist(\hat{s}_p, \hat{s}_q) > \sqrt{2l(1-\theta)}$ or $\langle q, \cdot\rangle \notin SL_p$.

Time Complexity. As mentioned earlier, inserting/removing a transformed subsequence into/from the kd-tree incurs $O(\log(w-l))$ time. Algorithm 1 requires at least $O(\log(w-l) + m_e)$ time, where $m_e = |SL_e| + |PL_e|$. Also, Algorithm 2 requires at least $O(\log(w-l) + m_n)$ time. Recall that m_n is the cardinality of returned (transformed) subsequences by Range-Search$(\hat{s}^\phi_n, \sqrt{2\phi(1-\theta)})$. If we compute the exact score of s_p, $O(l|PL_p|)$ time is required, since we need to scan PL_p and each Pearson correlation computation incurs $O(l)$ time. Let S' be a set of subsequences whose exact scores are computed when the window slides. The total time complexity of SRMM is $O(\log(w-l) + m_e + m_n + \sum_{S'} l|PL_p|)$. It is important to note that $|S'|$ is very small practically. For example, in our experiments, $|S'| \leq 1$ on average. If we consider a polylogarithmic factor, i.e., $\log(w-l)$, can be seen as a constant, the time complexity of SRMM is dependent only on the upper-bound scores of the expired and new subsequences in practice.

5 Experiment

This section introduces our experimental results. We evaluated SRMM and the baseline algorithm introduced in Sect. 2.2. All experiments were conducted on a PC with 3.4 GHz Core i7 CPU and 16 GB RAM, and all the algorithms were implemented in C++.

5.1 Setting

In the following setting, we measured the average update time per a slide of the window.

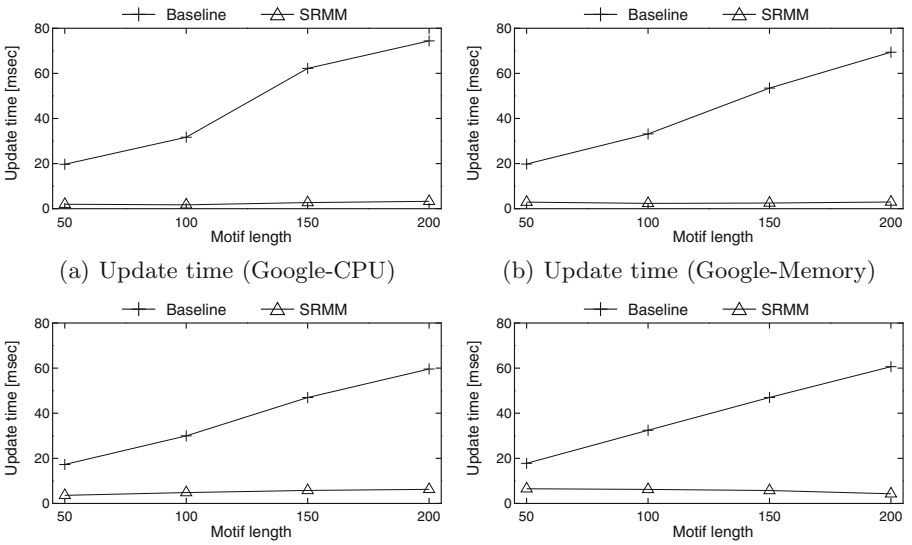
Datasets. We used four real datasets.

– Google-CPU [18]: this time-series is a merged sequence of CPU usage rate of machines in Google compute cells, and its length is 133,902.
– Google-Memory [18]: this time-series is a merged sequence of memory usage of machines in Google compute cells, and its length is 133,269.
– GreenHouseGas [12]: this is a time-series of green house gas concentrations with length 100,062.
– RefrigerationDevices[1]: this is a sequence of energy consumption of a refrigerator, and its length is 270,000.

Parameters. Table 1 summarizes the parameters used in the experiments and bold values are default values. We set $\phi = \frac{l}{2}$, and when we investigate the impact of a given parameter, the other parameters are fixed.

Table 1. Configuration of parameters

Parameter	Values
Motif length, l	50, **100**, 150, 200
Window-size, w [×1000]	5, **10**, 15, 20
Threshold, θ	0.75, 0.8, 0.85, **0.9**, 0.95



(a) Update time (Google-CPU)
(b) Update time (Google-Memory)
(c) Update time (GreenHouseGas)
(d) Update time (RefrigerationDevices)

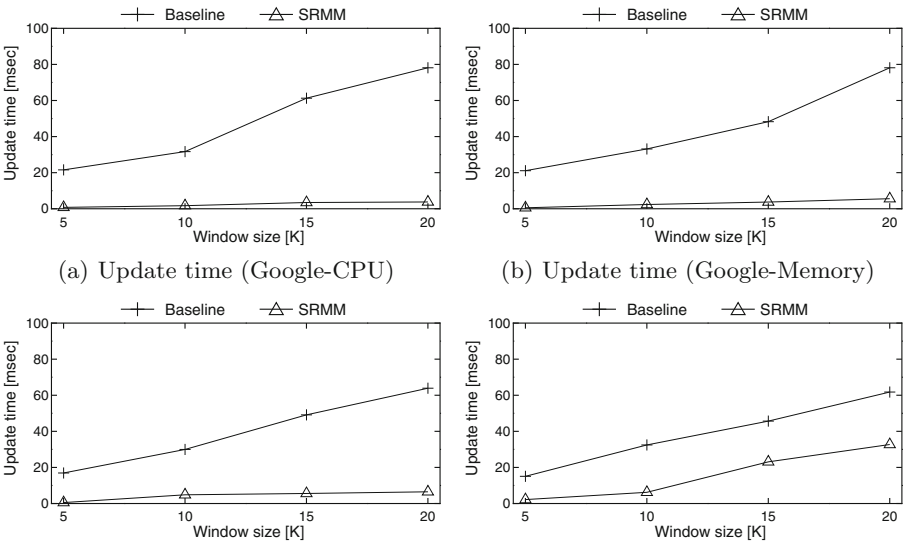
Fig. 3. Impact of l

[1] http://timeseriesclassification.com/index.php.

5.2 Result

Varying l. We first investigate the impact of motif length, and Fig. 3 shows the result. We see that the update time of the baseline algorithm linearly increases, as l increases. This is reasonable since its time complexity is $O((w-l)l)$. On the other hand, SRMM is not sensitive to l. As l increases, we need more time to compute Pearson correlation. However, for fixed θ, m_e and m_n decrease as l increases. For a large l, we tend to have a long distance between two subsequences, i.e., their Pearson correlation tends to be low. Hence, it becomes difficult for subsequences to be similar to other ones, which is the reason why m_e and m_n decrease. SRMM therefore has a stable performance even when l varies. This scalability is a good advantage against the baseline, and SRMM is up to 24.5 times faster than the baseline.

Varying w. We next investigate the impact of window size. As can be seen from Fig. 4, we have a very similar result to that in Fig. 3. The time complexity of the baseline is linear to w, so this result is also straightforward. A difference is that the update time of SRMM also increases. As w increases, the score of each subsequence tends to be larger, i.e., m_e and m_n become larger. SRMM therefore needs longer update time when w is large.

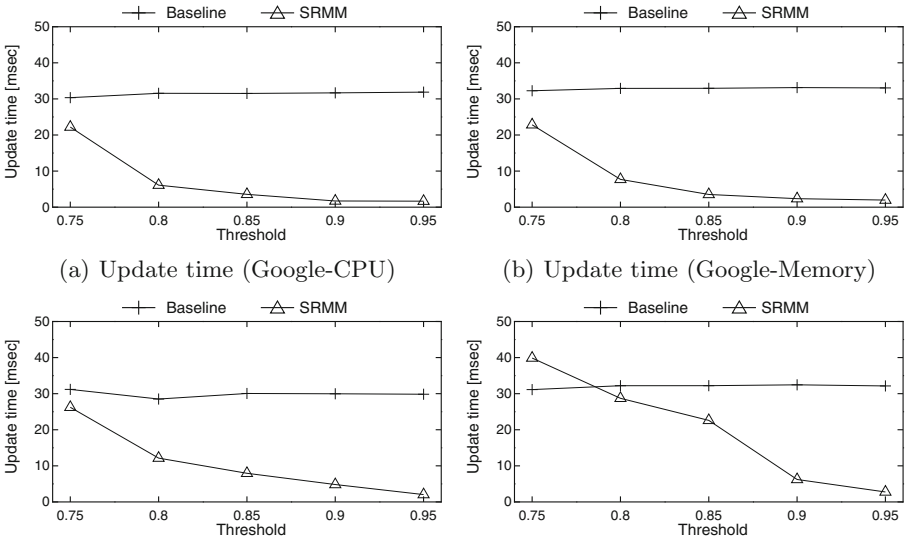


(a) Update time (Google-CPU) (b) Update time (Google-Memory)

(c) Update time (GreenHouseGas) (d) Update time (RefrigerationDevices)

Fig. 4. Impact of w

Varying θ. Finally, we report the impact of threshold, and the result is shown in Fig. 5. Because the baseline algorithm scans all subsequences in the window



(a) Update time (Google-CPU) (b) Update time (Google-Memory)

(c) Update time (GreenHouseGas) (d) Update time (RefrigerationDevices)

Fig. 5. Impact of θ

whenever the window slides, θ does not affect the performance of the baseline. On the other hand, the update time of SRMM decreases as θ increases. From Eq. (3), we see that the distance threshold becomes shorter as θ increases. Range queries in SRMM therefore report less subsequences. In other words, m_e and m_n also decrease, which provides the result in Fig. 5.

We can see that SRMM incurs longer update time than the baseline when $\theta = 0.75$. We observed that there are many similar subsequences for each subsequence in RefrigerationDevices when θ is small. In such cases, we cannot prune the exact score computation and the upper-bounding can be overhead. Note that many applications require a motif that has highly correlated subsequences, and as Figs. 5(a)–(d) show, SRMM can update the motif quite fast when θ is large.

6 Conclusion

Due to the trend that recent IoT-based applications generate streaming time-series, analyzing time-series in real-time becomes more important. This paper addressed the problem of monitoring a range motif (a subsequence which appears repetitively the most in a given time-series), for the first time. As an efficient solution to this problem. we proposed SRMM. This algorithm can avoid unnecessary score computation by exploiting Piecewise Approximate Aggregation and kd-tree. The results of our experiments using four real datasets show the efficiency and scalability of SRMM.

In this paper, we considered an one-dimensional time-series. Recently, a device is becoming to have multiple sensors and can generate a multi-dimensional

time-series. As a future work, we plan to address the range motif monitoring of a multi-dimensional streaming time-series.

Acknowledgement. This research is partially supported by JSPS Grant-in-Aid for Scientific Research (A) Grant Number JP26240013, JSPS Grant-in-Aid for Scientific Research (B) Grant Number JP17KT0082, and JSPS Grant-in-Aid for Young Scientists (B) Grant Number JP16K16056.

References

1. Begum, N., Keogh, E.: Rare time series motif discovery from unbounded streams. PVLDB **8**(2), 149–160 (2014)
2. Bentley, J.L.: Multidimensional binary search trees used for associative searching. Commun. ACM **18**(9), 509–517 (1975)
3. Castro, N., Azevedo, P.: Multiresolution motif discovery in time series. In: SDM, pp. 665–676 (2010)
4. Chen, Y., Nascimento, M.A., Ooi, B.C., Tung, A.K.: SpADe: on shape-based pattern detection in streaming time series. In: ICDE, pp. 786–795 (2007)
5. Chiu, B., Keogh, E., Lonardi, S.: Probabilistic discovery of time series motifs. In: KDD, pp. 493–498 (2003)
6. Grabocka, J., Schilling, N., Schmidt-Thieme, L.: Latent time-series motifs. TKDD **11**(1), 6 (2016)
7. Keogh, E., Chakrabarti, K., Pazzani, M., Mehrotra, S.: Dimensionality reduction for fast similarity search in large time series databases. KIS **3**(3), 263–286 (2001)
8. Lam, H.T., Pham, N.D., Calders, T.: Online discovery of top-k similar motifs in time series data. In: SDM, pp. 1004–1015 (2011)
9. Li, Y., Zou, L., Zhang, H., Zhao, D.: Computing longest increasing subsequences over sequential data streams. PVLDB **10**(3), 181–192 (2016)
10. Li, Y., Yiu, M.L., Gong, Z., et al.: Quick-motif: an efficient and scalable framework for exact motif discovery. In: ICDE, pp. 579–590 (2015)
11. Lin, J., Keogh, E., Wei, L., Lonardi, S.: Experiencing sax: a novel symbolic representation of time series. Data Min. Knowl. Disc. **15**(2), 107–144 (2007)
12. Lucas, D., et al.: Designing optimal greenhouse gas observing networks that consider performance and cost. Geosci. Instrum. Methods Data Syst. **4**(1), 121 (2015)
13. Moshtaghi, M., Leckie, C., Bezdek, J.C.: Online clustering of multivariate time-series. In: SDM, pp. 360–368 (2016)
14. Mueen, A., Keogh, E.: Online discovery and maintenance of time series motifs. In: KDD, pp. 1089–1098 (2010)
15. Mueen, A., Keogh, E., Zhu, Q., Cash, S., Westover, B.: Exact discovery of time series motifs. In: SDM, pp. 473–484 (2009)
16. Nguyen, H.L., Ng, W.K., Woon, Y.K.: Closed motifs for streaming time series classification. KIS **41**(1), 101–125 (2014)
17. Patel, P., Keogh, E., Lin, J., Lonardi, S.: Mining motifs in massive time series databases. In: ICDM, pp. 370–377 (2002)
18. Reiss, C., Wilkes, J., Hellerstein, J.L.: Google cluster-usage traces: format+ schema, pp. 1–14. Google Inc., White Paper (2011)
19. Shieh, J., Keogh, E.: i SAX: indexing and mining terabyte sized time series. In: KDD, pp. 623–631 (2008)

20. Yankov, D., Keogh, E., Medina, J., Chiu, B., Zordan, V.: Detecting time series motifs under uniform scaling. In: KDD, pp. 844–853 (2007)
21. Yeh, C.C.M., et al.: Matrix profile I: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets. In: ICDM, pp. 1317–1322 (2016)
22. Zhu, Y., et al.: Matrix profile II: exploiting a novel algorithm and GPUs to break the one hundred million barrier for time series motifs and joins. In: ICDM, pp. 739–748 (2016)