# Efficiently Mining Frequent Representative Motifs in Large Collections of Time Series

Stijn J. Rotman
*Department of CSAI*
*Tilburg University*
Tilburg, The Netherlands
s.j.rotman@tilburguniversity.edu

Boris Cule
*Department of CSAI*
*Tilburg University*
Tilburg, The Netherlands
b.cule@tilburguniversity.edu

Len Feremans
*Department of Computer Science*
*University of Antwerp*
Antwerp, Belgium
len.feremans@uantwerpen.be

***Abstract*—The discovery of repeated structures in time series, known as motifs, is an important data mining task. Various techniques exist to mine motifs in either a database of time series or within one or two individual time series, either for a user-defined motif length or a range of lengths. However, mining frequent motifs of variable length in large time series databases remains an unsolved task that is computationally expensive. We propose FRM-Miner, an efficient algorithm for discovering more informative patterns in time series data, i.e., motifs of different length that are non-overlapping, occur frequently and where the euclidean distance between the motif and its various occurrences is minimal. Unlike current state-of-the-art approaches, FRM-Miner can efficiently find variable motif lengths in large time series databases. Through extensive experimentation, we show desirable properties of FRM-Miner, such as robustness to noise and expressive power, thereby discovering motifs that remain undetected using state-of-the-art methods. Additionally, our method is highly scalable, taking only 2.95 hours to discover informative sets of motifs on all 128 time series data sets of the UCR Time Series Archive, where related state-of-the-art algorithms such as Ostinato require several days.**

## I. Introduction

Motif discovery in time series data has been an active research area for nearly two decades [1], with various approaches proposed to address the task [2]. Most algorithms for finding motifs that are based on the Matrix Profile focus on finding similar motif pairs within one or two time series [3]–[6]. Furthermore, Ostinato [7] and VACOMI [8] find consensus motifs that are consistently conserved across a collection of time series. Most of these algorithms require an exact motif length to be specified, but VALMOD [6] and VACOMI are able to mine motifs in a specified range of lengths as well.

In the current era of expansive data collection driven by technologies like the Internet of Things, vast amounts of user and sensor data are collected from numerous devices. The demand for scalable data mining grows. As time series grow, existing methods that rely on computing the distance between all segment pairs struggle and fail when applied to huge collections of time series. Moreover, existing methods are unable to discover frequent motifs, i.e., recurring patterns across numerous time series.

In this work, we introduce FRM-Miner, a novel approach for finding frequent motifs in large collections of time series. To accomplish this, we use techniques from the field of sequential pattern mining. As sequential pattern mining methodology requires discrete data, we use the Symbolic Aggregate approXimation (SAX) data representation [9] to transform the time series into discrete values. In order to illustrate our main contributions, the main steps of FRM-Miner are visualised in Fig. 1, using contours of cattle in the MPEG-7 data set [10].

Our main contributions can be summarised as follows:

- We propose to discover *representative* motifs which are frequently occurring *non-redundant* patterns in a set of time series.
- We find motifs of *varying length* from large collections of time series. We are particularly interested in such varied-length patterns as they are often encountered in real-world applications [8], [11].
- We develop a new algorithm by applying *sequential pattern mining* on discretised time series to identify sequence motifs. Unlike related methods, our algorithm translates identified patterns back to the original continuous time series domain in order to measure the distances between the motif and its occurrences.
- The outcome is a much more *efficient* algorithm than current state-of-the-art methods for discovering consensus motifs.

The remainder of this paper is outlined as follows. First, Section II introduces the necessary notation and definitions we use throughout the paper. Then, Section III reviews the existing related work. Thereafter, Section IV describes FRM-Miner in depth. In Section V, we experimentally evaluate our method and compare it to existing algorithms. Finally, we discuss our findings in Section VI and present our main conclusions in Section VII.

## II. Definitions and Notation

We start by introducing the used notation, defining the necessary concepts, and describing our problem setting. A *time series* $T$ is an ordered series of $n$ real-valued numbers

$$T = t_1, \ldots, t_n \in \mathbb{R} \tag{1}$$

and a *time series database* $D$ consists of $z$ time series $T^1, \ldots, T^z$.

Typically, time series represent real-valued measurements or sensor readings taken at regular time intervals. However, the

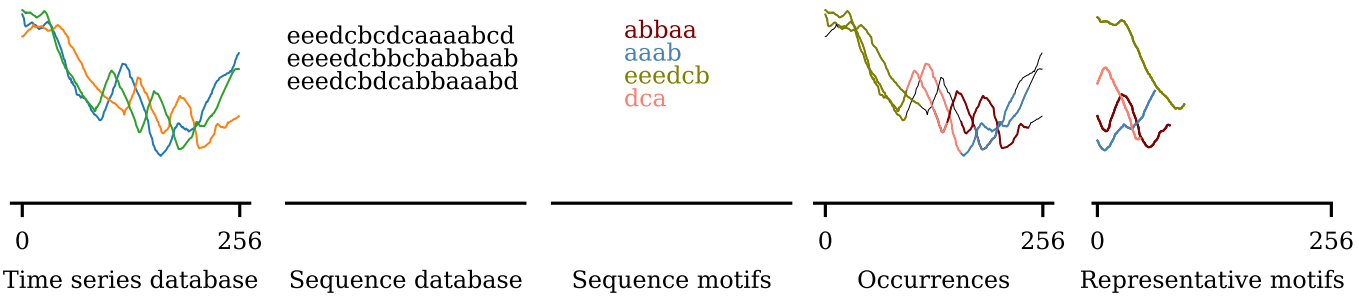


Fig. 1. Pipeline of all steps in FRM-Miner. The time series are discretised to sequences. Frequent sequence motifs are discovered and mapped back to time series occurrences, which are then used to construct representative motifs.

'time' dimension in time series does not necessarily always correspond to actual time, and can also be used to represent various kinds of other data, such as equidistant points in space. Note that individual time series in database $D$ may be of different or equal length.

A *subseries* of time series $T^v$ is denoted as $T^v_{j,l}$, with $1 \leq l \leq n$ and $1 \leq j \leq j+l-1 \leq n$. The subseries is the series of $l$ contiguous values in $T^v$ starting from position $j$, as

$$T^v_{j,l} = t^v_j, t^v_{j+1}, \ldots, t^v_{j+l-1}. \tag{2}$$

Since our goal is to discover time series motifs using sequential pattern mining techniques, we now define the necessary concepts from that field, too.

A *sequence* $S$ is an ordered series of $m$ discrete items

$$S = i_1, \ldots, i_m, \tag{3}$$

a *sequence database* $D_s$ consists of $z$ sequences $S^1, \ldots, S^z$.

A *subsequence* of sequence $S^v$ is the sequence of $k$ discrete items in $S^v$ starting from position $j$, with

$$S^v_{j,k} = i^v_j, i^v_{j+1}, \ldots, i^v_{j+k-1}, \tag{4}$$

where $1 \leq k \leq m$ and $1 \leq j \leq j+k-1 \leq n$.

A common way of moving from the real-valued time series to the discrete sequences domain is via the SAX representation. Given a time series $T^v$ of length $n$, a segment length $seglen$ and an alphabet size $\alpha$, *SAX* represents $T^v$ as a sequence $S^v$ with length $\lceil \frac{n}{seglen} \rceil$ that can take $\alpha$ different discrete values. The original SAX implementation [9] differs in some aspects. Section IV provides a detailed comparison between the original SAX representation and our own version.

As the sequences are created by discretising the time series, each sequence in $D_s$ will correspond to exactly one time series in $D$. Equally, each subsequence in an individual sequence $S^v$ will correspond to a subseries in time series $T^v$. We define this as the *corresponding subseries* $cs$ in $T^v$. Given a subsequence $S^v_{j,k}$ of $S^v$, the corresponding subseries is denoted $T^v_{j',k'}$, with $j' = (j-1) * seglen + 1$ and $k' = k \times seglen$.

We now move on to defining the patterns we aim to find. As a sequence, a *sequence motif* $sm$ is a series of discrete items

$$sm = i'_1, \ldots, i'_k, \tag{5}$$

where $k$ denotes its length. A sequence motif $sm$ *occurs* in sequence $S^v$ if there exists a position $j$ in $S$ such that $S^v_{j,k} = sm$.

The *support* of a sequence motif $sm$ in $D_s$, denoted as $sup(sm)$, is the fraction of sequences in $D_s$ in which the sequence motif occurs. Given a minimum support threshold *minsup*, a *frequent sequence motif* is any sequence motif $sm$ that satisfies

$$sup(sm) \geq minsup, \tag{6}$$

we use $\mathcal{P}$ to denote the set of all frequent sequence motifs.

It follows from the Apriori principle that all sequence motifs contained in a frequent sequence motif are frequent themselves [12]. This leads to the discovery of many overlapping sequence motifs. We define *overlap* as

$$overlap(sm_1, sm_2) = \frac{lcs(sm_1, sm_2)}{min(k_1, k_2)}, \tag{7}$$

where *lcs* refers to the longest common subsequence, *min* refers to the minimum of the values, and $k_r$ refers to the length of sequence motif $sm_r$. In Section IV, we explain how overlap is used to remove redundant sequence motifs.

Naturally, if a sequence motif $sm$ occurs in a sequence $S^v$, then it also occurs in the corresponding time series $T^v$. Moreover, if the occurrence of $sm$ in $S^v$ is subsequence $S^v_{j,k}$, then the motif occurrence in $T^v$ is the corresponding subseries $T^v_{j',k'}$. Note, however, that the support is defined as the fraction of time series that contain the motif, but that, in some cases, the motif may occur multiple times in a single time series. To handle such cases, we define motif representations that best capture such various occurrences.

Given $r$ occurrences $T^v_{j'_1,k'}, \ldots, T^v_{j'_r,k'}$ of a sequence motif $sm$ in time series $T^v$, the *average occurrence* $ao$ of $sm$ in $T^v$ is defined as the step-wise average of each element

$$ao^v = \mu(t^v_{j'_1}, \ldots, t^v_{j'_r}), \ldots, \mu(t^v_{j'_1+k'-1}, \ldots, t^v_{j'_r+k'-1}). \tag{8}$$

We are now ready to define the frequent representative time series motifs that we aim to discover.

Given a frequent sequence motif $sm$ and the set of time series $T^p, \ldots, T^q \in D$ in which $sm$ occurs, the *representative*

*motif* $rm$ of $sm$ in $D$ is defined as the step-wise average of the average occurrences of $sm$ in $T^p, \ldots, T^q$

$$rm = \mu(ao_1^p, \ldots, ao_1^q), \ldots, \mu(ao_{k'}^p, \ldots, ao_{k'}^q). \quad (9)$$

Next, we measure how well the representative motif is conserved in the time series that contain it. Given a representative motif $rm$ and a corresponding subseries $cs$, the *Euclidean Distance* is obtained by computing

$$ED(rm, cs) = \sqrt{\sum_{i=1}^{k'} (rm_i - cs_i)^2}, \quad (10)$$

taking the square root of the sum of squared step-wise differences between $rm$ and $cs$.

Given a representative motif $rm$ and a time series $T^v$, the Euclidean Distance for $rm$ in $T^v$ with corresponding subseries $cs_r^v, \ldots, cs_s^v$ is denoted as $ED_{rm}^v$ and computed as

$$ED_{rm}^v = min(ED(rm, cs_r^v), \ldots, ED(rm, cs_s^v)). \quad (11)$$

Lastly, we define the Normalised Average Euclidean Distance (*NAED*), which measures how well a representative motif $rm$ matches the occurrences within the different time series. Given a representative motif $rm$ of length $k'$ and the set of time series that contain the motif $T^p, \ldots, T^q$, $NAED_{rm}$ is defined as

$$NAED_{rm} = \frac{1}{k'} \mu(ED_{rm}^p, \ldots, ED_{rm}^q). \quad (12)$$

The NAED allows us to rank the discovered representative motifs on their ability to generalise. The normalisation of dividing by the length of the motif gives us the ability to compare motifs of different length.

## III. Related Work

Because we use methodology and concepts from motif discovery as well as sequential pattern mining, an overview of the relevant literature in both fields is provided. We start with an overview of existing work in time series motif discovery and continue with an overview of sequential pattern mining.

### A. Time Series Motif Discovery with SAX

Motif discovery in time series data has long been a topic of interest for researchers, with various approaches proposed to address the task. Patel et al. introduced the problem of motif discovery and proposed EMMA, an algorithm that uses a predecessor of SAX to speed up distance computations [1]. Fig. 2 gives a basic example of the SAX representation. Like many other algorithms, EMMA requires a user-specified motif length. Moreover, the algorithm is designed for finding motifs within a single time series, and not in large collections of time series. HIME also uses SAX and finds motifs by dividing the time series into smaller subseries and recursively discovering frequent patterns in each subseries [13]. Like EMMA, this algorithm is designed for finding motifs in a single time series. MrMotif uses an extension of SAX to reduce memory use when mining large time series for motifs [14].

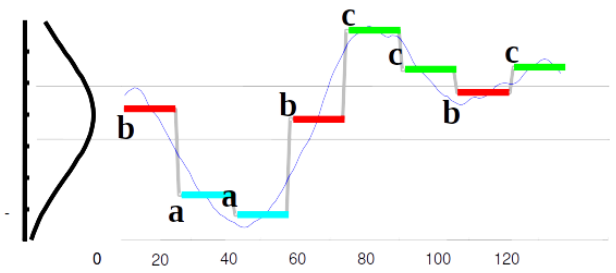


Fig. 2. SAX representation of a time series with $n = 128$ to the sequence *baabccbc* as illustrated by Lin et al. [9]. The time series is divided into $w$ segments, which are binned into $\alpha$ discrete values. Higher values for $w$ and $\alpha$ result in higher fidelity with respect to the original time series. In this case, $w = 8$ and $\alpha = 3$.

Stepping away from the task of motif discovery in time series, another approach that uses SAX for dimensionality reduction is the ULISSE algorithm [15]. This algorithm is aimed at finding subseries of variable length that correspond to a certain input query in large databases of time series. MiSTiCl also discretises time series data into SAX words, and employs string mining methodology to find discriminative features [16]. Likewise, PETSC uses variable-length sequential patterns as features for time series classification [11]. The still widespread use of SAX and its broad applicability make it a natural choice for FRM-Miner's discretisation step.

In summary, existing algorithms have their strengths in specific contexts, but they are not specifically tailored for discovering variable length motifs that frequently occur in large databases of time series. The limitations lie in their fixed-length motif assumptions and focus on different aspects of data mining tasks. Addressing the challenge of finding frequent variable length motifs in large time series databases remains an active area of research.

### B. Time Series Motif Discovery with the Matrix Profile

Nowadays, the Matrix Profile is a popular method for finding motif pairs, two subseries that are the most similar to each other, within one or two time series. Algorithms to compute the Matrix Profile include STAMP [3], an anytime algorithm, STOMP [4], which is not anytime but more efficient, and SCRIMP++ [5], which is both anytime and efficient. While this approach is useful in certain problem settings, it is limited to discovering motif pairs of a specified length and cannot be applied to larger collections of time series.

MOEN [17], HIME [13], and VALMOD [6] are able to find motifs of variable length, but are still restricted to finding these motifs in one or two time series. On the other hand, Ostinato [7] is able to work with larger databases of time series. However, it can only discover motifs of a specified length that occur in every time series. The same work proposes a *k of P* variant that finds the consensus motif in $k$ of the time series in a database with $P$ time series in total. This relaxes the requirement of the consensus motif occurring in every time series, though the search for $k$ gets increasingly expensive if $P$ gets larger. VACOMI [8] addresses some

of these limitations by efficiently performing variable length consensus motif discovery across multiple time series using a lower bound. However, it still requires that the motifs occur in every time series in the database, as no *k of P* variant exists.

Many of the existing motif discovery algorithms based on the Matrix Profile have a different focus than discovering motifs in collections of time series. This is no surprise, as the Matrix Profile is computed on a single time series or on pairs of time series. Algorithms that extend on the matrix profile to facilitate motif discovery have significant drawbacks as well. None of these algorithms address the task of discovering frequent motifs of variable length.

### C. Sequential Pattern Mining

GSP [12] and PrefixSpan [18] are two popular algorithms for mining sequential patterns in a data set. GSP employs a breadth-first search strategy with a pruning technique to reduce the search space. The two-step process generates candidate patterns of length $k$ by joining frequent patterns of length $k-1$ and prunes candidates with insufficient support. PrefixSpan is an extension of GSP, finding frequent sequential patterns with a prefix-suffix structure. It uses a depth-first search strategy combined with projected databases to reduce the search space and improve efficiency. The algorithm may use pseudo-projected databases to further optimise the search process. GSP is more suitable for dense data and PrefixSpan is more suitable for sparse data.

We propose a novel pattern mining algorithm that utilises insights of both GSP and PrefixSpan. In contrast to traditional sequential pattern mining with an unbounded number of gaps, we consider sequential motifs, or strings, where symbols must occur consecutively.

## IV. FRM-MINER

This section introduces the algorithm that is used for mining frequent representative motifs of variable length. After creating discrete sequences from the time series using SAX, a novel sequential pattern mining algorithm finds sequence motifs in the sequence database. Redundant sequence motifs are removed and the representative time series motifs are constructed from the sequence motifs, after which the representative motifs are ordered on their NAED values. The above is succinctly captured in the pseudocode in Fig. 3.

This algorithm takes a database of time series $D$, minimum support *minsup* used for sequence motif mining, segment length *seglen*, and alphabet size $\alpha$ for discretisation. Additionally, a minimum and maximum pattern length *lmin* and *lmax* and maximum pattern overlap *omax* can be set for efficiency or removing redundant patterns. Throughout the rest of this paper, we set $lmin = 3$, $lmax = \infty$, and $omax = 0.9$. The following sections give implementations of MINE_SEQUENCE_MOTIFS, REMOVE_REDUNDANT, and MAP, which are called in the pseudocode.

### A. Discrete Representation

SAX is a technique used to transform a time series into a sequence of discrete symbols [9]. The first step in this process involves representing the time series using Piecewise Aggregate Approximation (PAA). In SAX, the number of segments, denoted by the parameter $w$, is user-defined, as is alphabet size $\alpha$, which determines the number of discrete symbols each segment can be binned to. Lin et al. [9] propose a local binning strategy where normalisation and PAA are applied to each time series window separately. In contrast, we set the window size to the entire time series and apply normalisation over the entire time series.

```
1:  function MINE(D, minsup, seglen, α, lmin, lmax, omax)
2:      D_s ← SAX(D, seglen, α)
3:      P ← MINE_SEQUENCE_MOTIFS(D_s, minsup, lmax)
4:      sms ← REMOVE_REDUNDANT(P, lmin, omax)
5:      motifs ← ∅
6:      for all sm ∈ sms do
7:          rm, naed_rm ← MAP(D, sm, seglen)
8:          append (rm, naed_rm) to motifs
9:      end for
10:     sort motifs on naed
11:     return motifs
12: end function
```

Fig. 3. Frequent Representative Motif Miner

If the length of the time series is not evenly divisible by $w$, the original definition of SAX changes the weights of the observations in the time series to still discretise the time series into $w$ segments. For this reason, some elements in the time series may be weighted into two frames by SAX instead of one. This would reduce the similarity of the discrete representations to the repeated subseries, and therefore hinder the ability to find motifs. To address this issue, we introduce the parameter *seglen*, which denotes the number of elements in the segments, instead of $w$. In the case that the time series is not evenly divisible by *seglen*, all but the last segment consist of *seglen* items. The last segment then consists of the remaining values in the time series.

In the example from Fig. 2, the time series is of length 128 and is transformed into a sequence of length 8, the *seglen* is thus 16. The time series are divided into *seglen* sized segments, with the final segment consisting of the remaining elements. This ensures that individual symbols in the resulting discrete representation correspond to subseries of equal length in the original time series.

### B. Sequence Motif Mining

We introduce a novel approach for finding sequence motifs specifically tailored to mining without gaps (i.e., unlike traditional sequential pattern mining, we are only interested in patterns consisting of consecutive symbols in the input sequences). We propose the strategy of generating candidate patterns and pruning infrequent patterns by maintaining a map of pattern position indexes. The pseudocode for our sequence motif mining strategy is laid out in Fig. 4.

One scan is made over sequence database $D_s$ to initialise the index map with sequence motifs of length 1. Sequence motifs

```
1: function MINE_SEQUENCE_MOTIFS(D_s, minsup, lmax)
2:     Record indexes of all 1-patterns in index_map
3:     k ← 2
4:     while index_map[k − 1] ≠ ∅ ∧ k ≤ lmax do
5:         index_map[k] ← ∅
6:         C ← GET_CANDIDATES(index_map[k − 1])
7:         for all candidate ∈ C do
8:             Record indexes of candidate
9:             if sup(candidate) < minsup then
10:                Remove candidate from index_map
11:            end if
12:        end for
13:        k++
14:    end while
15:    return P
16: end function
```

```
1: function GET_CANDIDATES(index_map[k − 1])
2:     C ← ∅
3:     for all sm^1 ∈ index_map[k − 1] do
4:         for all sm^2 ∈ index_map[k − 1] do
5:             if sm^1_2, ..., sm^1_k = sm^2_1, ..., sm^2_{k−1} then
6:                 append sm^1_1, ..., sm^1_k, sm^2_k to C
7:             end if
8:         end for
9:     end for
10:    return C
11: end function
```

Fig. 4. Mine frequent sequence motifs

```
1: function REMOVE_REDUNDANT(P, lmin, omax)
2:     sort P in decreasing sm length
3:     for all sm_1 ∈ P do
4:         if LEN(sm_1) ≤ lmin then
5:             remove sm_1 from P
6:             continue
7:         end if
8:         for all sm_2 ∈ P with LEN(sm_2) < LEN(sm_1) do
9:             if OVERLAP(sm_1, sm_2) > omax then
10:                remove sm_2 from P
11:            end if
12:        end for
13:    end for
14:    return P
15: end function
```

Fig. 5. Remove redundant sequence motifs

that do not comply to a user-defined *minsup* are deleted and the algorithm continues to mine longer motifs. Each step in the mining process consists of creating candidate motifs, extending the index map with occurrences, and removing infrequent motifs. This process terminates if no frequent sequence motifs of a certain length are found or the candidates exceed *lmax*.

After index map initialisation, $D_s$ is not needed anymore, as its contents are ordered in the index map. Determining the occurrences of a sequence motif requires only the index map entries of its parents.

### C. Index Map

When finding sequence motifs in the sequence database, we need to be able to store occurrences in the sequences, and to later find their corresponding subseries. We achieve this via an index map data structure, that keeps track of the starting indexes of occurrences in each sequence. The index map is constructed as a nested structure of three levels. The first level contains the sequence motifs, the second level contains the indexes of sequences in $D_s$ with at least one occurrence of the sequence motif, and the third level contains the starting indexes of each occurrence. The index map data structure allows for time- and space-efficient storage and lookup of sequence motif occurrences and their corresponding subseries.

Suppose the index map contains two sequence motifs *abc* and *bca*. The candidate generated from these sequence motifs is *abca*. Recording where *abca* occurs in the database only requires comparing the entries in the index map under *abc* and *bca*, as each time *abc* and *bca* have subsequent starting indexes in a sequence, *abca* must occur. After recording the occurrences of *abca* in this manner, the support of the sequence motif can be found by looking at the length of the second level of the index map.

### D. Removing Redundant Motifs

After all frequent sequence motifs have been mined using the method laid out above, we need to remove irrelevant sequence motifs. We consider a motif redundant in two cases: if the motif is too short or if the motif is too similar to another frequent motif that is longer. Redundant motifs are removed according to the strategy laid out in the pseudocode in Fig. 5.

With the *lmin* parameter, users can set the minimum subsequence length to be considered interesting, the authors suggest setting this at 3 to conservatively only remove the most trivial short sequence motifs. As the mining algorithm generates candidates in a bottom-up manner, mining sequence motifs that are too short cannot be avoided. However, removing them is trivial.

The second way in which sequence motifs can be redundant, is if they are mostly contained in other sequence motifs. The level of overlap allowed can be set with the *omax* parameter. We suggest a value of 0.9 to remove patterns that are too similar. To illustrate, given sequence motifs *aaaaaa* and *aabaa*, their longest common subsequence is four and the length of the shortest sequence motif is five. The overlap, computed as the quotient of the longest common subsequence and the length of the shortest sequence motif, is thus 0.8. Since longer motifs are more informative than shorter motifs, if two motifs greatly overlap, we keep the longer one and remove the shorter one.

### E. Construction of Representative Motifs

The pseudocode in Fig. 6 describes how the index map is used to get a sequence motif's corresponding subseries, average occurrences, and representative motif. This procedure

**function** MAP($D, sm, seglen$)
  **for all** $v, indexes \in sm$ **do**
    **for all** $(j, k) \in indexes$ **do**
      $j' \leftarrow (j-1) \times seglen + 1$, $k' \leftarrow k \times seglen$
      $cs_w^v \leftarrow t_{j'}^v, \ldots, t_{j'+k'-1}^v$
    **end for**
    $ao^v \leftarrow$ step-wise average of $cs_r^v, \ldots, cs_s^v$
  **end for**
  $rm \leftarrow$ step-wise average of $ao^p, \ldots, ao^q$
  **for all** $cs^v \in \{cs^p, \ldots, cs^q\}$ **do**
    $ED_{rm}^v \leftarrow$ MIN(ED$(rm, cs_r^v), \ldots,$ED$(rm, cs_s^v)$)
  **end for**
  $NAED_{rm} \leftarrow \frac{1}{k'}\mu(ED_{rm}^p, \ldots, ED_{rm}^q)$
  **return** $rm$, $NAED_{rm}$
**end function**

Fig. 6. Map sequence motif to representative motif

loops over all indexes $j$ and lengths $k$ of pattern occurrences and maps them back to the original time series to construct corresponding subseries $cs_w^v$ for occurrence $w$ in time series $v$. Occurrences are stored in separate lists for each time series the pattern occurred in. The final representative motif is then constructed by taking the average of the occurrences for each time series separately, after which the average of these averages is taken. If a time series is not evenly divisible by *seglen* and the last symbol of the discrete representation is part of a sequence motif, the occurrence in the original time series can be shorter than the other occurrences. To account for this, we append NaN values to the occurrence, these values are not taken into account when constructing the average occurrences. In the case that the only occurrence of a sequence motif falls at the end of such a time series, the last values of an average occurrence can be NaN values as well, these are not taken into account when constructing the representative motif either. Similarly, if all average occurrences end with NaN values, the length of the representative motif is reduced to only contain numeric values.

After the representative motif is constructed, it can be evaluated by computing how well it fits the corresponding subseries using the NAED. Equation (12) in Section II is equal to the calculation in the pseudocode in Fig. 6. $cs^p, \ldots, cs^q$ denote the collections of corresponding subseries in $T^p, \ldots, T^q$. Each of these collections $cs^v$ contains $cs_r^v, \ldots, cs_s^v$. The calculation of $NAED_{rm}$ then averages and length-normalises each $ED_{rm}^v$ in $ED_{rm}^p, \ldots, ED_{rm}^q$.

### F. Computational Complexity

Sequential pattern mining has a theoretical worst-case time complexity that is exponential, i.e., given $\alpha$ bins there could be $O(\alpha^n)$ candidate sequential patterns up to length $n$. In contrast to sequential pattern mining, we search for frequent sequence motifs where the number of possible candidate motifs is limited by each sequence $S$, i.e., there are $|S| - 1$ candidate strings of length 1, $|S| - 2$ candidate strings of length 2, ... and $|S| - n + 1$ candidate strings of length $n$ occurring in each sequence $S$. Hence, the worst-case time complexity is quadratic in the length of each discrete sequence, i.e., $O(|D| \cdot |S|^2)$. We remark that, in practice, very long patterns are extremely rare [19]. Assuming a maximum length of $k$ on frequent sequence motifs, the worst-case time complexity becomes $O(k \cdot |D| \cdot |S|)$. We conclude that the proposed approach for motif mining thereby restricting the search space and pruning on *minsup* is only linear in the length of each sequence.

The time complexity of discretisation is linear in the database size $|D|$ and sequence length $|S|$. Note that the discrete sequences are shorter than the original time series, i.e. $|S| = \lceil |T|/seglen \rceil$, with $T$ the corresponding time series. The time complexity for removing patterns that are redundant, and reducing the amount of representative motifs that need to be constructed, is $O(|\mathcal{P}|^2)$, where we use $\mathcal{P}$ to denote the set of frequent sequence motifs.

For computing the representative motifs we compute the NAED based on all occurrences. A naive approach to compute occurrences would have a time complexity of $O(|D| \cdot |S| \cdot |\mathcal{P}|)$. However, we compute the occurrences of a sequence motif using the index map data structure that organises these occurrences in a nested manner. Hence, this search is linear in the number of occurrences for each sequence motif, i.e., $O(|D| \cdot |\mathcal{P}|)$. Finally, the calculation of the NAED of each representative motif is linear in the number of occurrences as well. In practice, we observe that all three steps, i.e., sequence motif mining, removing redundant motifs and mapping to representative motifs each require about a third of the total run time and that each run completes within a few minutes on most data sets.

## V. EXPERIMENTS

In this section, we demonstrate the viability and usefulness of our FRM-Miner algorithm. First, we assess FRM-Miner's ability to find motifs in noisy settings. Then, a run time experiment on benchmark data is performed. Subsequently, the impact of database size and time series length on performance is measured. We continue with an experiment assessing the impact of key parameters on performance and number of motifs found. Finally, we demonstrate real-world applicability of FRM-Miner in two case studies.

We provide a C++ implementation of FRM-Miner, which we have interfaced to Python using pybind11 [20]. For comparisons against Ostinato, we use the Python implementation provided in the Stumpy library [21]. This comparison is fair because both use a Python interface to call compiled code. The pyscamp library [22] uses a faster strategy to calculate the matrix profile, but contains no implementation of Ostinato. Our efforts of implementing Ostinato with the matrix profile calculations done by pyscamp did not result in a more efficient implementation of Ostinato overall. All experiments are run on a Linux Mint 21 machine with an AMD Ryzen 7 3800X processor. Our data and code are available at https://github.com/steenrotsman/frm-miner.

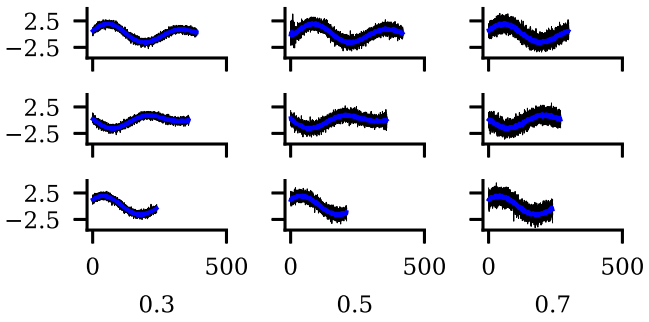


Fig. 7. Top 3 found motifs for 3 noise levels. Motif injected into 100% of time series. Each column corresponds to the level of noise in the axis label and contains the top motifs for that noise level.

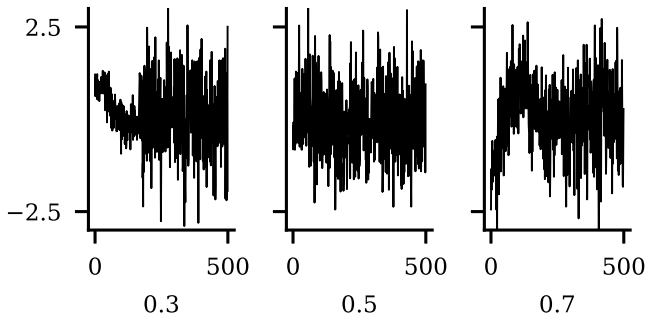


Fig. 9. Consensus motifs found by Ostinato for 3 noise levels shown on the axis labels. Motif injected into 75% of time series. The overlap with the injected motif is substantially smaller than that of the representative motifs.

### A. Robustness to noise

We first ask *to what extent is FRM-Miner able to find motifs in noisy settings?* To explore this, a database of 100 time series with 10 000 randomly generated observations each is generated from a Gaussian distribution. A constructed motif, consisting of 500 observations, is then inserted into each time series at a random location, with Gaussian noise added to the motif. The standard deviation in the Gaussian distribution is set at 0.3, 0.5, and 0.7. Throughout this experiment, we set $minsup = 0.3$, $seglen = 30$, and $\alpha = 4$.

Fig. 7 shows the resulting top 3 motifs for each noise level. Each column in the figure corresponds to a different level of noise added to the embedded motifs. The blue lines show the representative motifs, while the black lines around them show the best matching occurrences of those motifs in each time series that contain them.

From the figure, it becomes apparent that lower levels of noise lead to more comprehensible patterns. As the levels of noise increase, the motifs that are found tend to get shorter, though the top 3 motifs still cover most of the injected motif. In this setting, Ostinato correctly finds the motif for each noise level, but only when provided with the exact motif length in advance, whereas FRM-Miner determines the length of the motif automatically.

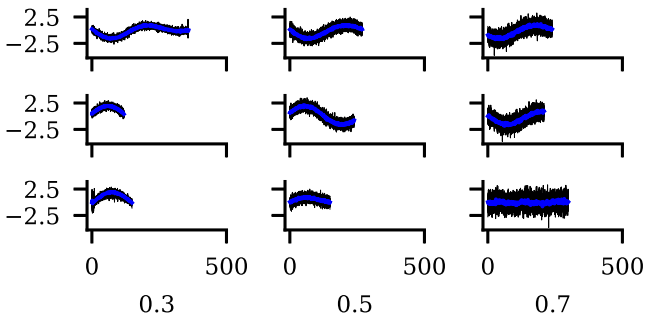


Fig. 8. Top 3 found motifs for 3 noise levels. Motif injected into 75% of time series. Each column corresponds to the level of noise in the axis label and contains the top motifs for that noise level.

We further increase the noisiness of the data by embedding the motif frequently, but not in all time series. Fig. 8 shows the top 3 motifs for each noise level after injecting the constructed motif in 75% of time series. In this setting, the entire motif is still recovered for the lowest noise level. At higher levels of noise, the recovered motifs are shorter than the corresponding motifs in Fig. 7.

Fig. 9 shows that Ostinato performs much worse in retrieving the motif in this setting. For the highest and lowest noise levels, the consensus motifs partially overlap with the injected motifs, but less than those found by FRM-Miner. Note, too, that Ostinato returns a particular individual occurrence of the discovered motif, including the noise, while our representative motif is obtained by averaging all the occurrences, resulting in a much smoother representation of the original motif (blue lines in Figures 7 and 8).

These experiments demonstrate our ability to find motifs in data that contains substantial amounts of noise. As long as the motif frequency remains compliant with the user-defined *minsup*, the time series that contain no instances have no impact on the ability to discover meaningful motifs.

### B. Run times compared to Ostinato

We continue our analysis of FRM-Miner by asking *how does FRM-Miner's run time performance compare to Ostinato?* Total run time on the UCR Time Series Classification Archive [23] is used as basis for comparison, because of the varying sizes and ratios between numbers of time series and time series lengths in the data sets.

We choose to compare against Ostinato because existing motif discovery algorithms aimed at finding motif pairs in one or two time series have entirely different problem settings. Furthermore, Ostinato and VACOMI have identical output given the same (range of) length(s).

FRM-Miner is run with *seglen* = 1, $\alpha$ = 5, and *minsup* = 0.3. We set *seglen* and *minsup* fairly low to demonstrate performance with adverse settings. The only parameter of Ostinato is the length of the consensus motif. We set the desired length at 10 percent of the shortest time series in a

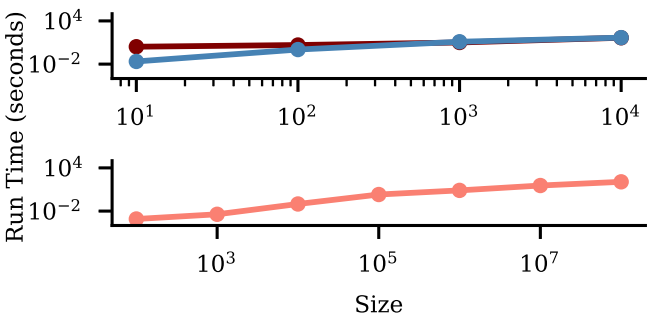


Fig. 10. Run times for different simulated data sizes. We observe an approximately linear relationship for time series quantity (*red*), length (*blue*), and total database size (*pink*), measured as number of data points.

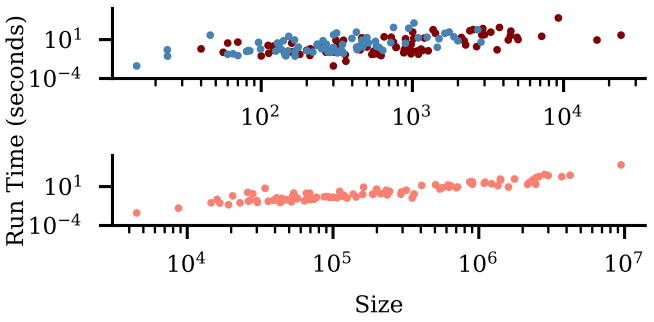


Fig. 11. Run times for different data sizes in the UCR archive. We observe an approximately linear relationship for time series quantity (*red*), length (*blue*), and total database size (*pink*), measured as number of data points.

data set, with the constraints that the length must be at least 3 and at most the length of the shortest time series.

The time required for FRM-Miner to analyse the entire archive is 2.95 hours. The Stumpy implementation of Ostinato requires 650.59 hours. Thus, FRM-Miner is multiple orders of magnitude faster than Ostinato. This quick execution time of FRM-Miner is indicative of its ability to efficiently process large amounts of data. A pure Python implementation of FRM-Miner requires 14.84 hours to complete, which is still notably faster than Ostinato.

Several factors contribute to Ostinato's inefficiency, including its inherent algorithmic complexity and emphasis on consensus motif discovery, requiring the presence of motifs in every time series within a data set. This constraint necessitates cross-comparison across all time series to identify common patterns, resulting in a significantly larger search space and increased execution time.

### C. Scalability of FRM-Miner

Next, we ask *how does data size affect run times for FRM-Miner?* To assess scalability, we conducted experiments with synthetic data sets, introducing Gaussian noise and motifs while altering time series quantity and length from 10 to 10 000. For each value of time series quantity, the average run times across the time series lengths is recorded and vice versa. The results for all combinations that lead to a certain total database size are averaged as well. The results, depicted in Fig. 10, reveal a linear relationship between size and run time. Each combination of time series quantity and length is evaluated 10 times to mitigate randomness in system noise. There is a minimal difference between the impact of increasing numbers of time series and time series lengths, which diminishes at greater sizes.

To validate these findings, we compared them with the run times from Section V-B. For each data set in the UCR time series archive, we record the time series quantity, average time series length, and total data set size, plotting these values against the run times in Fig. 11. Though the ranges of the data set sizes are smaller than in our simulated experiment, the congruent trends in Fig. 11 provide further support for FRM-Miner's scalability.

### D. Parameter impact

We continue the assessment of FRM-Miner's performance by asking *what is the impact of different parameter settings on run times?* To assess performance on data sets larger than those in the UCR archive, we collected daily volume data on all stocks with 100 or more records available on StockAnalysis.com. The resulting 5 862 time series range in length from 100 to 15 413, totalling 21 857 301 observations.

We run FRM-Miner on this time series database, varying the values for the core parameters: *minsup* is varied over 0.1, 0.3, 0.5, and 0.7, *seglen* is varied over 5, 10, 20, and 50, and $\alpha$ is varied over 3, 5, 7, and 9.

Fig. 12 shows the average run times and average number of discovered motifs for each parameter value. The outcomes for each value are obtained by averaging the results varying the other two parameters.

From the figure, we can see negative relationships between the average run times and all three parameters. This effect is stronger for *minsup* and especially *seglen* than for $\alpha$. The number of patterns found is decreasing in *minsup* and *seglen* as well. However, $\alpha$ shows a weak positive relationship with the number of discovered motifs.

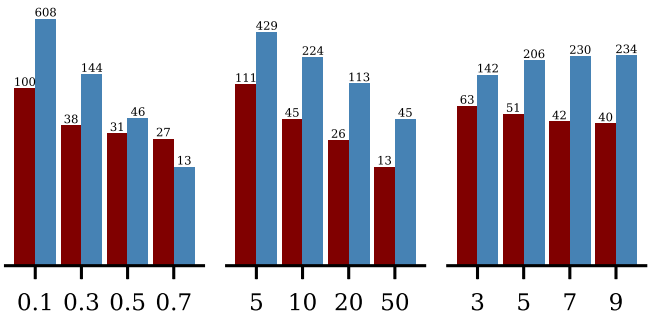


Fig. 12. Average run time in seconds (*red*) and number of discovered motifs (*blue*) for different values of *minsup* (*left*), *seglen* (*mid*), and $\alpha$ (*right*). Values are rounded to the nearest integer and plotted on a shared log scale.

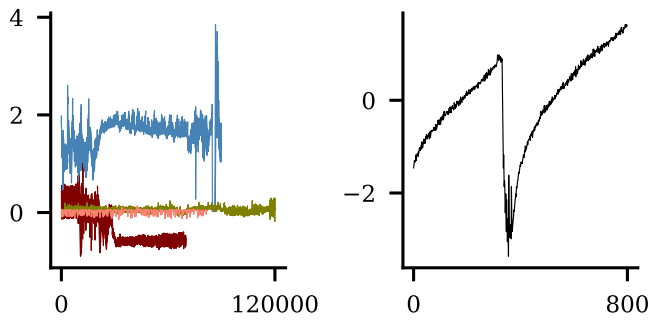


Fig. 13. Four EPG telemetry recordings of Asian citrus psyllid behaviour (*left*). Data set taken from Kamgar et al. [7]. Consensus motif of length 800 (*right*).

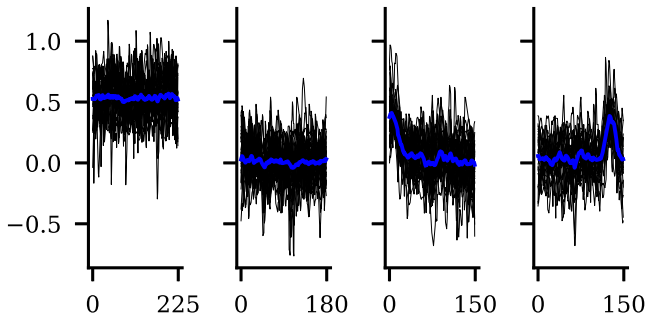


Fig. 15. Frequent motifs of variable length in cycling speed data. Parameter settings are *minsup* = 0.3, *seglen* = 15, and $\alpha$ = 5.

### *E. Use Case: Insect EPG Telemetry*

Let us consider the particularly noisy data set in Fig. 13. Ostinato finds a well-conserved consensus motif of length 800 [7]. In this experiment, we put to the test whether FRM-Miner is able to find a similar motif and if the constraints of a consensus motif obscure other interesting motifs. Even after (global) z-normalisation, the consensus motif occurs at different levels in each time series, because some of the time series are non-stationary due to having non-constant means. To mitigate this, we apply FRM-Miner not on the raw data, but on the first difference of the time series to make them stationary. After finding the best conserved representative motifs on the differenced data, we apply the indexes of its occurrences to the raw time series to find the representative motifs shown in Fig. 14. The motif that results from setting $minsup = 1$ closely resembles the consensus motif. Relaxing the *minsup* from 1 to 0.5, we find another well-conserved pattern that occurs in two of the four time series.

### *F. Use Case: Cycling speeds*

To illustrate real world applicability of FRM-Miner, we mine frequent motifs on data collected from the speed in 81 distinct cycling training sessions of one athlete with a GPS bike computer. The bike computer records the speed each second and the training sessions range from 680 seconds (11.33 minutes) to 30 629 seconds (8.51 hours), the database contains 549 981 observations in total.

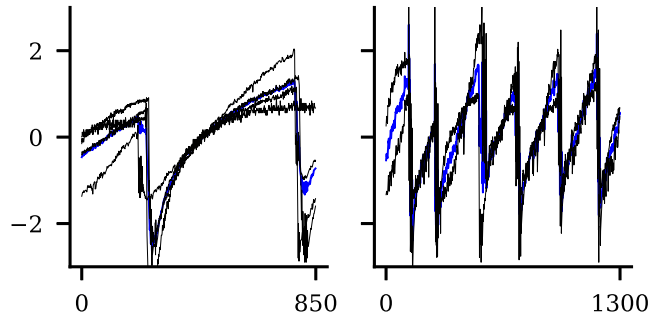


Fig. 14. Representative motif and motif occurrences for insect telemetry. Parameter settings are *minsup* = 1 (left), *minsup* = 0.5 (right), *seglen* = 50, $\alpha$ = 4. The left representative motif corresponds to the motif found by Ostinato, the right motif was previously unknown.

Fig. 15 shows the top 4 patterns found with fairly standard parameter settings. From the figure, some patterns in the cycling behaviour of the athlete can be distinguished. The first two motifs unveil a tendency toward sustaining steady speeds during workouts, with the first motif surpassing the average and the second motif aligning closely with it. The third and fourth motifs are characterised by short surges in speed. These occurrences potentially signal intriguing behaviour that can serve as a basis for further analysis.

## VI. Discussion

FRM-Miner operates with three core parameters: *minsup*, *seglen*, and $\alpha$. Their settings are adaptable based on the data characteristics. Because of the demonstrated efficiency of FRM-Miner, experimentation with different parameter settings is very fast and can be performed interactively. For our experiments, we found *minsup*=0.3 consistently effective. For *seglen*, we recommend aligning with the data's time intervals. For instance, intervals of 15, 30, or 60 could suit data collected each minute, capturing motifs at quarter, half, and full-hour marks. The alphabet's size also hinges on data specifics. We suggest moderate values like 4, 5, or 6, maintaining a balance between SAX's discretisation ability and the ability to distinguish different values. Beyond the core parameters, optional fine-tuning is available using *lmin*, *lmax*, and *omax*. We propose *lmin* = 3, *lmax* = $\infty$, and *omax* = 0.9.

While the discrete SAX representation gives FRM-Miner the ability to find frequent motifs efficiently, it does lead to a loss in accuracy through segmentation and binning. Multiple instances of the same frequent motif in the continuous domain could have different discrete representations. Consequently, two identical sequence motifs could have more dissimilar corresponding subseries than two different sequence motifs. Thus, FRM-Miner cannot be guaranteed to find all frequent representative motifs satisfying a certain similarity threshold. This drawback can be mitigated by experimenting with different parameter settings (e.g., smaller segments or more bins)

If finding the exact number of matches of a motif in a time series database is a priority, the frequent motifs discovered with FRM-Miner can be used as seed sequences for further analysis, for instance using MASS [3]. Other directions of future work include exploring the usefulness of the retrieved motifs for tasks such as clustering, classification, and anomaly detection and motif discovery in multivariate data sets.

## VII. Conclusion

This work proposes FRM-Miner, an algorithm for the discovery of frequent representative motifs in large databases of time series. By leveraging sequential pattern mining techniques, we are able to efficiently discover frequent time series motifs of varying lengths. This process is enhanced by a redundancy elimination method that ensures that the output contains only the most interesting motifs. Subsequently, these motifs are ranked based on their capacity to generalise raw occurrences within the original time series data. Our experimental evaluation underscores the algorithm's scalability, precision, and the quality of the discovered motifs, highlighting its real-world relevance.

In contrast to most state-of-the-art methods, FRM-miner enables mining motifs of varying length on large data sets. This is an important innovation, as motif length is typically hard to know in advance. We achieve this by introducing three core parameters. Through systematic experimentation, we offer insights into the impacts of various parameter configurations and provide recommendations for parameter values that can serve as a starting point to further experimentation.

Notably, the efficiency inherent to our algorithm enables fast exploration across an array of parameter settings. This property makes FRM-Miner especially suitable for scenarios demanding rapid search and discovery. The ability to quickly experiment with different parameter settings reduces the drawback of having three parameters to tune. In conclusion, our study not only advances the field of time series motif discovery through FRM-Miner but also contributes to the broader understanding of scalable and efficient algorithmic exploration in the domain of data mining and pattern recognition.

## References


[1] P. Patel, E. Keogh, J. Lin, and S. Lonardi, "Mining motifs in massive time series databases," in *2002 IEEE International Conference on Data Mining, 2002. Proceedings.* IEEE, 2002, pp. 370–377.

[2] S. Torkamani and V. Lohweg, "Survey on time series motif discovery," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 7, no. 2, p. e1199, 2017.

[3] C.-C. M. Yeh, Y. Zhu, L. Ulanova, N. Begum, Y. Ding, H. A. Dau, D. F. Silva, A. Mueen, and E. Keogh, "Matrix profile i: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets," in *2016 IEEE 16th international conference on data mining (ICDM)*. Ieee, 2016, pp. 1317–1322.

[4] Y. Zhu, Z. Zimmerman, N. S. Senobari, C.-C. M. Yeh, G. Funning, A. Mueen, P. Brisk, and E. Keogh, "Matrix profile ii: Exploiting a novel algorithm and gpus to break the one hundred million barrier for time series motifs and joins," in *2016 IEEE 16th international conference on data mining (ICDM)*. IEEE, 2016, pp. 739–748.

[5] Y. Zhu, C.-C. M. Yeh, Z. Zimmerman, K. Kamgar, and E. Keogh, "Matrix profile xi: Scrimp++: time series motif discovery at interactive speeds," in *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2018, pp. 837–846.

[6] M. Linardi, Y. Zhu, T. Palpanas, and E. Keogh, "Matrix profile goes mad: variable-length motif and discord discovery in data series," *Data Mining and Knowledge Discovery*, vol. 34, no. 4, pp. 1022–1071, 2020.

[7] K. Kamgar, S. Gharghabi, and E. Keogh, "Matrix profile xv: Exploiting time series consensus motifs to find structure in time series sets," in *2019 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2019, pp. 1156–1161.

[8] M. Zhang, P. Wang, and W. Wang, "Efficient consensus motif discovery of all lengths in multiple time series," in *International Conference on Database Systems for Advanced Applications*. Springer, 2022, pp. 540–555.

[9] J. Lin, E. Keogh, L. Wei, and S. Lonardi, "Experiencing sax: a novel symbolic representation of time series," *Data Mining and knowledge discovery*, vol. 15, pp. 107–144, 2007.

[10] L. J. Latecki, R. Lakamper, and T. Eckhardt, "Shape descriptors for non-rigid shapes with a single closed contour," in *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No. PR00662)*, vol. 1. IEEE, 2000, pp. 424–429.

[11] L. Feremans, B. Cule, and B. Goethals, "Petsc: pattern-based embedding for time series classification," *Data Mining and Knowledge Discovery*, vol. 36, no. 3, pp. 1015–1061, 2022.

[12] R. Srikant and R. Agrawal, "Mining sequential patterns: Generalizations and performance improvements," in *International conference on extending database technology*. Springer, 1996, pp. 1–17.

[13] Y. Gao and J. Lin, "Hime: discovering variable-length motifs in large-scale time series," *Knowledge and Information Systems*, vol. 61, pp. 513–542, 2019.

[14] N. Castro and P. Azevedo, "Multiresolution motif discovery in time series," in *Proceedings of the 2010 SIAM international conference on data mining*. SIAM, 2010, pp. 665–676.

[15] M. Linardi and T. Palpanas, "Scalable data series subsequence matching with ulisse," *The VLDB Journal*, vol. 29, no. 6, pp. 1449–1474, 2020.

[16] A. Raza and S. Kramer, "Accelerating pattern-based time series classification: a linear time and space string mining approach," *Knowledge and Information Systems*, vol. 62, no. 3, pp. 1113–1141, 2020.

[17] A. Mueen and N. Chavoshi, "Enumeration of time series motifs of all lengths," *Knowledge and Information Systems*, vol. 45, no. 1, pp. 105–132, 2015.

[18] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, "Mining sequential patterns by pattern-growth: The prefixspan approach," *IEEE Transactions on knowledge and data engineering*, vol. 16, no. 11, pp. 1424–1440, 2004.

[19] T. Calders, C. Garboni, and B. Goethals, "Approximation of frequentness probability of itemsets in uncertain data," in *2010 IEEE International Conference on Data Mining*. IEEE, 2010, pp. 749–754.

[20] W. Jakob, J. Rhinelander, and D. Moldovan, "pybind11 – seamless operability between c++11 and python," 2017, https://github.com/pybind/pybind11.

[21] S. M. Law, "STUMPY: A Powerful and Scalable Python Library for Time Series Data Mining," *The Journal of Open Source Software*, vol. 4, no. 39, p. 1504, 2019.

[22] Z. Zimmerman, K. Kamgar, N. S. Senobari, B. Crites, G. Funning, P. Brisk, and E. Keogh, "Matrix profile xiv: scaling time series motif discovery with gpus to break a quintillion pairwise comparisons a day and beyond," in *Proceedings of the ACM Symposium on Cloud Computing*, 2019, pp. 74–86.

[23] H. A. Dau, E. Keogh, K. Kamgar, C.-C. M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratanamahatana, Yanping, B. Hu, N. Begum, A. Bagnall, A. Mueen, G. Batista, and Hexagon-ML, "The ucr time series classification archive," October 2018, https://www.cs.ucr.edu/~eamonn/time_series_data_2018/.