International Journal of Data Science and Analytics (2026) 22:220
https://doi.org/10.1007/s41060-026-01181-y

REGULAR PAPER

FRM-Miner: efficient motif discovery in large collections of time series

Stijn J. Rotman[1] · **Boris Cule**[1] · **Len Feremans**[2]

Received: 22 September 2025 / Accepted: 1 June 2026

Abstract

The discovery of repeated structures in time series, known as motifs, is an important data mining task. Various techniques exist to discover motifs within a single time series or a pair of time series, either for a user-defined motif length or a range of lengths. However, discovering motifs in larger collections of time series is given less attention to. We propose an improved version of FRM-Miner, an efficient algorithm for discovering informative patterns in collections of time series. FRM-Miner converts time series with SAX and applies frequent sequential pattern mining to the resulting symbolic sequences, after which frequent patterns are mapped back to time series occurrences. FRM-Miner discovers non-overlapping motifs that occur frequently throughout the time series database. Discovered motifs are ranked on the z-normalised euclidean distance between their occurrences. Unlike current state-of-the-art approaches, FRM-Miner finds frequent motifs sets with varying lengths and levels of support efficiently in large time series databases. We compare run time efficiency and memory requirements between different versions of the algorithm. Through extensive experimentation, we demonstrate the robustness to noise, real-world applicability, and scalability of FRM-Miner.

Keywords Time series · Motif discovery · Sequential pattern mining

1 Introduction

Motif discovery in time series data has been an active research area for over two decades [28], with various approaches proposed to address the task [38]. Most motif discovery research focuses on finding motifs pairs (for example, with the Matrix Profile [44]) or motif sets [43] in one or two time series. Most algorithms for motif discovery require an exact motif length to be specified, but VALMOD [25] can find motif pairs in a specified range of lengths and the Pan Matrix Profile [26] generalises the Matrix Profile to all possible subsequence lengths. Furthermore, LoCoMotif [43] discovers motif sets that have varying lengths.

The methods mentioned above are not designed for time series databases. Yet large collections of time series naturally arise in applications where multiple entities are monitored simultaneously. For example, industrial monitoring involves fleets of sensors tracking many assets, requiring joint analysis to detect recurring patterns or failures [21, 35]. In wearable and biomedical settings, longitudinal data from multiple subjects (e.g., heart rate or activity) must be analysed to identify shared temporal patterns [1, 34, 41]. Similarly, traffic and energy networks generate distributed streams that require system-wide analysis [2, 18]. In such settings, methods designed for single time series or pairs of time series become restrictive, as they do not scale or capture patterns across collections. This motivates the need for methods that discover patterns across time series databases rather than analysing each series in isolation.

Kamgar et al. [17] propose Ostinato, which finds *consensus motifs*, motifs that occur in every time series in a time series database. The same work defines the $k\ of\ P$ consensus motif, which is the best consensus motif in k of the P time

A preliminary version of this paper was published under the title "Efficiently Mining Frequent Representative Motifs in Large Collections of Time Series" at BigData 2023 [31]. Len Feremans is funded by Research Fund Flanders, Grant ID 12B0V24N.

✉ Stijn J. Rotman
s.j.rotman@tilburguniversity.edu

Boris Cule
b.cule@tilburguniversity.edu

Len Feremans
len.feremans@uantwerpen.be

[1] Department of Intelligent Systems, Tilburg University, Tilburg, The Netherlands

[2] Data Science Institute, UHasselt, Diepenbeek, Belgium

Published online: 06 July 2026

series in the database. However, no algorithm that addresses this problem setting is provided.

While useful in contexts where exact solutions are needed, Ostinato has several drawbacks: a single consensus motif occurrence is returned, the length of the consensus motif needs to be known in advance, and the consensus motif must occur in every time series. Several other works address these limitations and propose extensions of Ostinato. Variable-length consensus motif discovery is made possible with VACOMI [45], which iterates over different lengths more efficiently than Ostinato. De Paepe and Van Hoecke [10] propose Anytime Ostinato, which is able to rank all subseries, as opposed to just finding the top 1 motif as Ostinato does.

We present FRM-Miner, a novel approach for finding *frequent motifs* of variable length in large collections of time series. Like consensus motifs, frequent motifs occur across time series in a database. However, we loosen the constraint of the motif occurring in *every* time series, requiring motifs to be present in a user-defined minimum number of time series instead.

To illustrate our main contributions, the main steps of FRM-Miner are visualised in Fig. 1. The first column shows our problem setting: a collection of time series. The second column shows the intermediate discrete representation. The third column contains the sequence motifs: discrete patterns that are frequent in the sequence database. The fourth column maps the sequence motifs back to the continuous time series space. Finally, the fifth column shows the resulting representative motif.

Our contribution over most existing algorithms for motif discovery is that we find motifs in time series *databases* as opposed to single time series or time series pairs. Our contributions over Ostinato and its extensions are as follows:

- We loosen the requirement of the consensus motif for a motif to occur in every time series. Instead, FRM-Miner finds all motifs that comply with a user-defined minimum threshold. This addresses the variety of time series in big data settings, as the presence of time series that do not contain the consensus motif of other time series are to be expected [10].
- We find frequent motifs of *varying length* from large collections of time series, automatically determining the optimal length for each motif separately. We are particularly interested in such varied-length patterns as they are often encountered in real-world applications [12, 45].
- Instead of returning a single occurrence, as the existing algorithms do, we propose to discover *representative* motifs or motif *sets* instead. Representative motifs summarise the motif occurrences in the database.
- FRM-Miner is better suited for settings where the volume of data is high. As we show in Sect. 5, it scales well if number of time series, time series length or both increase. With growth in data quantity, growth in noise can be expected, which we show FRM-Miner is able to handle better than Ostinato.

To accomplish this, we use sequential pattern mining techniques. As sequential pattern mining methodology requires discrete data, we use the Symbolic Aggregate approXimation (SAX) data representation [23] to transform the time series into discrete values.

This paper extends the work by Rotman et al. [31] that was published and presented at the IEEE BigData'23 conference. Apart from a thorough rewrite and a more extensive experimental evaluation, the main additional contributions we make are as follows. First, we allow more flexibility in the lengths of discovered motifs, removing noisy start and end regions from occurrences. This results in the ability to find motifs of all lengths, as opposed to just multiples of $seglen$ (a parameter used in the discretisation step). Second, we perform an analysis of the *space* complexity of FRM-Miner in Sect. 4, additional to the existing analysis of time complexity. Third, we introduce a much more memory-efficient algorithm, which can also be found in Sect. 4.

The remainder of this paper is organised as follows. First, Sect. 2 introduces the necessary notation and definitions we use throughout the paper. Then, Sect. 3 reviews the existing related work. Thereafter, Sect. 4 describes FRM-Miner in depth. In Sect. 5, we experimentally evaluate our method and compare it to existing algorithms. Finally, we discuss our findings in Sect. 6.

2 Definitions and notation

In this section, we introduce the necessary notation and formally define the problem setting.

A *time series* T is an ordered series of n real-valued numbers

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

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

A *time series database* D is a set of z time series

$$D = T^1, \ldots, T^z, \tag{2}$$

the individual time series in database D may be of different or equal length. We remark that the case of $z = 2$ can be

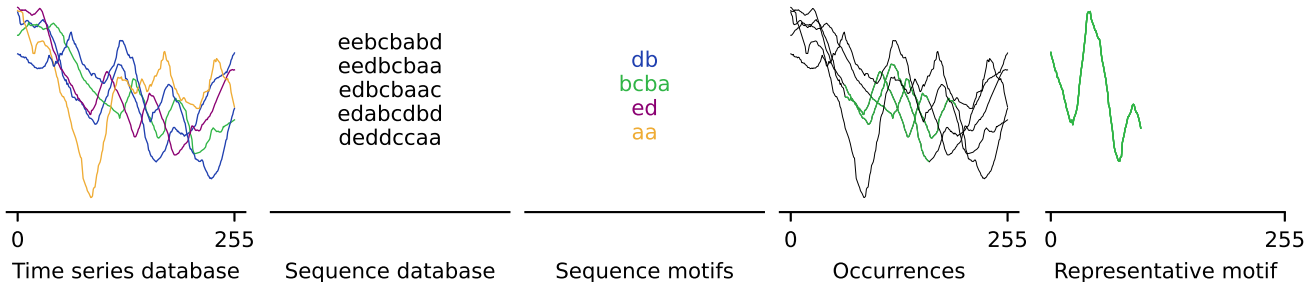

Fig. 1 Succinct pipeline of the steps in FRM-Miner. The time series are discretised to sequences. Frequent sequence motifs are discovered and mapped back to time series occurrences, which are then used to construct representative motifs. We show only the occurrences and representative motif for one motif, but the same steps are applied to all motifs. The time series are derived from images of cattle in the MPEG-7 data set [19]. For each image, the contour is divided into 256 points. For each point, the distance to the middle of the image is calculated, thus forming a time series

analysed with conventional methods that use the Matrix Profile calculated on an ABJoin of the two time series. However, we typically assume the number of time series to be much larger, which is a problem setting not suitable for the vanilla Matrix Profile.

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, i.e.,

$$T^v_{j,l} = t^v_j, t^v_{j+1}, \ldots, t^v_{j+l-1}. \tag{3}$$

We now move on to defining the patterns we aim to find.

A *motif* is defined as a collection of up to one subseries of a certain length per time series. Though we aim to discover motifs of several lengths, each motif's occurrences must have the same length. The *support* of a motif set is defined as the fraction of time series in the database with a subseries in the motif. A *frequent motif set* is a motif set with a support equal to or greater than a user-defined support threshold, $minsup$.

Motifs can be compared and ordered based on their distance.

Considering two time series (or subseries) T^a and T^b of the same length n as points in an n-dimensional space, the distance between the time series is calculated with the z-normalised Euclidean distance (z-ED):

$$z\text{-}ED(T^a, T^b) = \sqrt{\sum_{i=1}^{n} \left(\frac{T^a_i - \mu^a}{\sigma^a} - \frac{T^b_i - \mu^b}{\sigma^b} \right)^2}, \tag{4}$$

where μ and σ denote the mean and standard deviation of the respective time series.

The distance is used to rank the discovered representative motifs. A key feature of our proposed methodology is that the discovered motifs can have different lengths. As argued by Linardi et al. [25]: "Since we can *discover* motifs of different lengths, we also need to be able to *rank* motifs of different lengths." The same work proposes to length-normalise the Euclidean distance by dividing by the square root of the length of the motif. This allows the comparison of frequent representative motifs of different length without bias towards longer motifs.

Given the subseries T^a and T^b of length n, their length-normalised distance d is given by

$$d(T^a, T^b) = \frac{z\text{-}ED(T^a, T^b)}{\sqrt{n}}. \tag{5}$$

The *extent* of a set of subseries $S = \{T^a_{p,l}, \ldots, T^b_{q,l}\}$ is calculated as the maximal pairwise distance between two subseries:

$$extent(S) = \max_{(T^a, T^b) \in S \times S} (d(T^a, T^b)). \tag{6}$$

Closely related to the definition of extent, is that of *radius*, given by Kamgar et al. [17]. The *radius* of a subseries $T^a_{j,l}$ is the maximum distance between that subseries and each nearest neighbour in D:

$$radius(T^a_{j,l}, D) = \max_{T^b \in D} \left(\min_{i \in [0, |T^b| - l]} d(T^a_{j,l}, T^b_{i,l}) \right). \tag{7}$$

The consensus motif is then defined as the subseries with the smallest radius. We remark that the definition of fixed-length consensus motifs is not robust to outliers. For instance, if a single time series in D is generated using a different underlying process, a clear consensus motif does not exist and the best consensus motif has a too high radius making if difficult to distinguish from noise.

Our aim is to identify the frequent motif sets present in D. In Sect. 4, we present an algorithm that discovers frequent motif sets.

3 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 work in time series motif discovery and continue with an overview of sequential pattern mining.

3.1 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 [28]. 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. Sequitur [22], GrammarViz [36], and HIME [14] also use SAX for variable-length motif discovery and MrMotif uses an extension of SAX to reduce memory use when discovering motifs in long time series [7]. Like EMMA, these algorithms are designed for finding motifs in single time series. Extending them to find frequent motifs in time series databases instead of single time series is non-trivial.

Stepping away from the task of motif discovery in time series, another approach that uses SAX for dimensionality reduction is the ULISSE algorithm [24]. 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 [30]. Likewise, PETSC uses variable-length sequential patterns as features for time series classification [12]. The still widespread use of SAX and its broad applicability make it a natural choice for FRM-Miner's discretisation step.

3.2 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. The Matrix Profile can be computed with the algorithms STAMP [44], which is anytime, STOMP [48], which is more efficient but not anytime, and SCRIMP++ [47], which is both anytime and efficient. While the Matrix Profile approach is useful in certain problem settings, it can only discover motif pairs of a specified length and cannot be applied directly to larger collections of time series.

Motiflets [33], LoCoMotif [43] and MotiPlus [11] find motif sets, which can have more occurrences than motif pairs. However their problem settings are limited to single time series.

MOEN [27], MDTW_WedgeTree [39, 40], HIME [14] and the earlier mentioned LoCoMotif are able to find motifs of variable length, as well as VALMOD [25] and an improvement over VALMOD that requires just a single computation of the distance matrix [42]. The Pan Matrix Profile is a time series primitive that calculates the Matrix Profile for each subseries length and can thus be used for motif discovery of all lengths as well [26]. Each of these algorithms is still restricted to discovering motifs in one or two time series.

On the other hand, Ostinato [17] 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 would relax the requirement of the consensus motif occurring in every time series, but no algorithmic definition to discover $k\ of\ P$ consensus motifs is given. VACOMI [45] addresses some of Ostinato's limitations by speeding up the enumeration of user-defined lengths. VACOMI achieves this speed-up over sequential calls to Ostinato by using information collected at shorter lengths. The information collected at shorter lengths is used to form a lower bound on the Euclidean distance of subsequent lengths. De Paepe and Van Hoecke [10] extend the typical problem setting of Ostinato of finding the best consensus motif with a variant that ranks all subseries and can thus discover consensus motifs beyond the top 1.

In summary, many of the existing motif discovery algorithms based on the Matrix Profile have a different focus than discovering motifs in collections of time series, aiming to discover *motif pairs*. This is no surprise, as the Matrix Profile is computed on a single time series or on pairs of time series. Algorithms that build on the matrix profile to facilitate motif discovery in larger databases of time series have significant drawbacks as well. None of these algorithms address the task of discovering frequent motifs of variable length in a large collection of time series.

3.3 Sequential pattern mining

Sequential pattern mining is a task of data mining and pattern recognition that aims to discover interesting sequential patterns from discrete sequential data.

Several algorithmic approaches have been proposed to solve the task of sequential pattern mining. The Apriori-based algorithm GSP (Generalized Sequential Pattern) [37]

performs a breadth-first search to mine frequent sequential patterns. It has a two-step process that first generates candidate patterns of length k by joining frequent patterns of length $k-1$ and then prunes candidates with insufficient support.

Contrarily, PrefixSpan [29] performs a depth-first search. This algorithm is an extension of GSP that finds frequent sequential patterns with a prefix-suffix structure. That is, the depth-first search strategy is combined with a projected database to reduce the search space and improve efficiency. Pseudo-projection is used instead of projection to not duplicate storage, which further optimises the search process. Typically, GSP is more suitable for dense data (that is, discrete databases with many frequent sequential patterns) and PrefixSpan is more suitable for sparse data (discrete databases with few frequent sequential patterns).

Both algorithms operate on the principle that every subpattern of a frequent sequential pattern must be frequent itself. That is, the set of frequent patterns contains all subpatterns of these frequent patterns as well. For this reason, frequent pattern mining typically produces many redundant patterns, which has resulted in the development of definitions that retain only informative patterns. The set of closed patterns contains all frequent patterns that are not contained in another frequent pattern with the same support [15]. Furthermore, the set of maximal patterns captures only the most extensive sequences of events or items that are frequent and cannot be extended without decreasing their support [13]. In other words, any frequent pattern that is fully contained in another frequent pattern is removed from the set. We base the redundancy elimination step of FRM-Miner on these ideas.

These definitions and algorithms are aimed at the discovery of sequential patterns in databases of discrete sequences. As described above, a database of time series can be transformed into a sequence database using SAX. With this discrete representation, sequential pattern mining algorithms can be used to extract knowledge from time series databases as well. While this can produce valuable insights, the extracted patterns remain in the discrete domain. In contrast, our goal is to find motifs, which are in the continuous time series domain. The typical definition of a sequential pattern does not just include frequently occurring substrings. Instead, sequential patterns can contain *gaps* where any item can occur in between the elements of the pattern. As we map the sequential patterns that we discover back to the time series database to find continuous motifs, the interpretability might be hindered by permitting gaps in the patterns. Therefore, in contrast to traditional sequential pattern mining with an unbounded number of gaps, we only consider sequential motifs where symbols occur consecutively. We propose a novel pattern mining algorithm that uses insights from GSP as well as from PrefixSpan.

4 FRM-Miner

This section introduces our proposed algorithm 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 ranked on their distances. The above is succinctly captured in Algorithm 1.

Algorithm 1 Frequent Representative Motif Miner

```
1: function MINE(D, minsup, seglen, α, omax, δ)
2:     D_δ ← δ^th difference of D
3:     D_s ← SAX(D_δ, seglen, α)
4:     P ← MINE_SEQUENCE_MOTIFS(D_s, minsup)
5:     sms ← REMOVE_REDUNDANT(P, omax)
6:     motifs ← ∅
7:     for all sm ∈ sms do
8:         rm, d_rm ← MAP(D, sm, seglen)
9:         append (rm, d_rm) to motifs
10:    end for
11:    sort motifs on d_rm
12:    return motifs
13: end function
```

This algorithm takes a database of time series D, a minimum support threshold $minsup \in [0, 1]$ used for sequence motif mining, and a segment length $seglen \in \mathbb{N}$ and an alphabet size $\alpha \in \mathbb{N}$ for discretisation. Additionally, a maximum pattern overlap $omax \in [0, 1]$ can be set for removing redundant patterns and a degree of differencing δ can be set to discover motifs in derivatives of the original time series. Lines 2 and 3 apply the preprocessing steps of differencing and applying SAX. Discrete sequential patterns are mined and filtered in lines 4 and 5 by calling MINE_SEQUENCE_MOTIFS and REMOVE_REDUNDANT. Lines 6–10 then process each sequence motif to a time series motif in MAP. Finally, line 11 sorts the discovered motifs based on their length-normalised distance. The following sections provide details of the procedures called in the MINE procedure in Algorithm 1.

The pipeline of steps FRM-Miner applies is shown in more detail in Fig. 2. The time series are discretised in the top row. The middle row shows the discovered frequent sequence motifs and the removal of redundant motifs. For all non-redundant frequent sequence motifs, the rest of the steps are the same: the occurrences of the sequences are mapped to time series occurrences. For each time series, an average occurrence is constructed. The average occurrences are then combined to construct a representative motif.

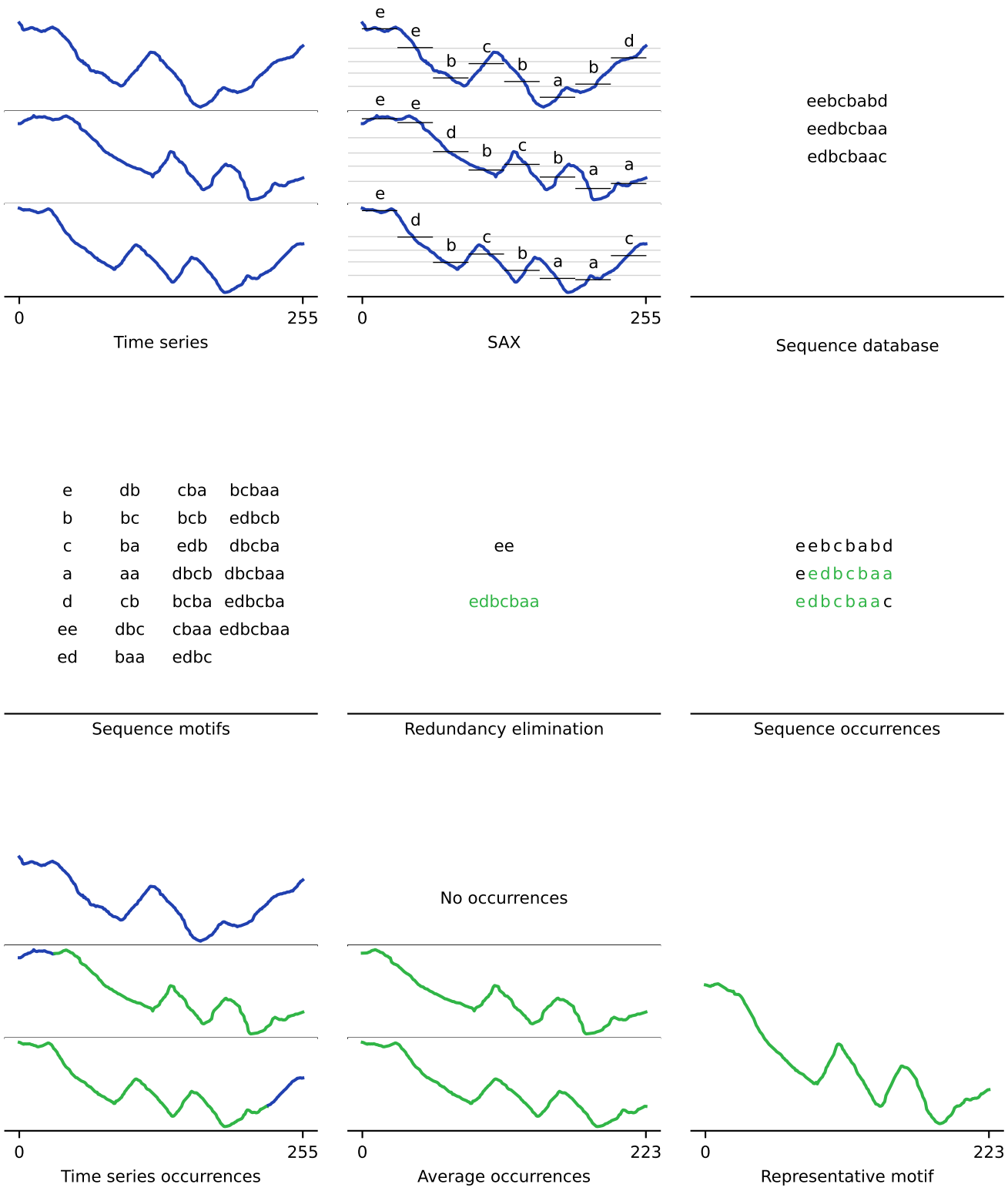

Fig. 2 Detailed pipeline of the steps in FRM-Miner. We show only the steps from a sequence motif to a representative motif for a single motif (highlighted in green). However, the same steps are applied to all sequence motifs. The time series are the same as those shown in Fig. 1, but the time series database contains just the first 3 time series (color figure online)

4.1 Discrete representation

SAX is a technique used to represent a time series as a sequence of discrete symbols [23]. 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 α, which determines the number of discrete symbols each segment can be binned to.

The original definition of SAX proposes a local binning strategy where z-normalisation and PAA are applied over a sliding window. That is, each time series is divided into equal-length windows. Each window is z-normalised separately and then converted to discrete symbols. However, separately z-normalising subseries would require setting another parameter (the subseries length), which would amplify discretisation error and could cause FRM-Miner to miss frequent motif occurrences.

The main reason for applying SAX over a sliding window of a time series is to handle concept drift in a time series. However, handling such non-stationarities in time series can be done in other ways, as we discuss in Sect. 5.4. Therefore, we set the window size to the entire time series and apply normalisation over the entire time series.

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 a 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.

Figure 3 presents an example of a time series represented as a sequence after applying SAX. The time series is of length 256 and is transformed into a sequence of length 8, the $seglen$ is thus 32. 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.

4.2 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 first present our original algorithm [31]. Then, we present an improvement of the algorithm aiming to reduce memory usage. We compare the two approaches in terms of time and space complexity in Sect. 4.5. The two approaches are experimentally compared in Sect. 5.

4.2.1 Original approach

We propose the strategy of generating candidate patterns and pruning infrequent patterns by maintaining a map of pattern position indices. Algorithm 2 lays out the original approach to mining sequence motifs.

Algorithm 2 Mine frequent sequence motifs (FRM-Miner 1.0)

```
1: function MINE_SEQUENCE_MOTIFS(D_s, minsup)
2:     Record indices (u, j) of all 1-patterns in index_map[1]
3:     k ← 2
4:     while index_map[k − 1] ≠ ∅ do
5:         index_map[k] ← ∅
6:         C ← GET_CANDIDATES(index_map[k − 1])
7:         for all c ∈ C do
8:             Record indices of c
9:             if sup(c) < minsup then
10:                Remove c 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, sm^2) ∈ index_map[k − 1]^2 do
4:         if sm^1_2, ..., sm^1_k = sm^2_1, ..., sm^2_{k−1} then
5:             append sm^1_1, ..., sm^1_k, sm^2_k to C
6:         end if
7:     end for
8:     return C
9: end function
```

One scan is made over sequence database D_s to initialise the index map with sequence motifs of length 1 (line 2). Sequence motifs that do not comply to a user-defined $minsup$ are deleted and the algorithm continues to mine longer motifs, starting with a length of 2 (line 3). Each step in the mining process consists of creating candidate motifs (line 6; GET_CANDIDATES function), extending the index map with occurrences (line 8), and removing infrequent motifs (lines 9–10). This process terminates if no frequent sequence motifs of a certain length are found (line 4). Finally, the set of frequent sequence motifs, $\mathcal{P}$, is returned.

The GET_CANDIDATES function initialises the set of candidates of a certain length, $\mathcal{C}$, to the empty set (line 2). It then calculates the Cartesian product of the set of length $k-1$ sequence motifs and loops over each pair (line 3). For each

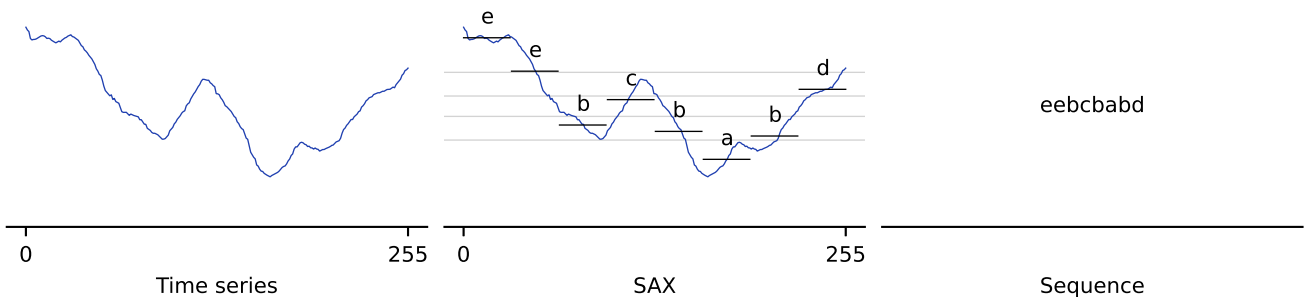

Fig. 3 Pipeline illustrating the SAX representation. The time series is divided into w segments of length $seglen$, which are binned into α discrete values. In this case, $w = 8$, $seglen = 32$, and $\alpha = 5$

pair, it is checked if the last $k-1$ elements of sequence motif 1 are equal to the first $k-1$ elements of sequence motif 2 (line 4). If this is the case, the sequence motifs are combined into a new candidate motif, which is then added to the set of candidate motifs (line 5). Finally, the set of candidate sequence motifs of length k, $\mathcal{C}$ is returned.

4.2.2 Index map

When finding sequence motifs in the sequence database, we need to store occurrences of the sequences to find their corresponding subseries in later steps. We achieve this via an index map data structure, that keeps track of the starting indices of occurrences in each sequence. The index map is constructed as a nested data structure that contains three levels. The first level contains the sequence motifs, the second level contains the indices of sequences in D_s with at least one occurrence of the sequence motif, and the third level contains the starting indices 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 indices in a sequence, *abca* must occur. After recording the occurrences of *abca* in this manner, the support of the sequence motif is found by looking at the number of time series for which indices have been recorded in the index map. Moreover, as the sequence motifs contained in a frequent sequence motif must be frequent themselves, we only consider patterns that are in the index map for candidate generation.

4.2.3 More efficient search

While the index map greatly increases mining efficiency, limiting the needed number of database scans to one, it does come with a drawback. Because subseries of sequence motifs are frequent themselves, storing the indices of sequence motif occurrences leads to the same indices being duplicated over the index map, causing high memory use. As an illustrative example, consider the sequence motif *aa*. If this sequence motif is frequent, so is its parent *a*. The starting indices of the two patterns are thus duplicated, as is illustrated in this representation of the index map:

```
{
    a: {0: [5], 1: [6, 7]},
    aa: {1: [6]}
}
```

Algorithm 3 Mine frequent sequence motifs (FRM-Miner 2.0)

1: **function** MINE_SEQUENCE_MOTIFS(D_s, $minsup$)
2: Record indices (u, j) of all 1-patterns in $index\_map[1]$
3: $k \leftarrow 2$
4: **while** $index\_map[k-1] \neq \emptyset$ **do**
5: $index\_map[k] \leftarrow \emptyset$
6: **for all** $(parent, u, j) \in index\_map[k-1]$ **do**
7: $candidate \leftarrow S^u_j \dots S^u_{j+k-1}$
8: **if** $candidate_1 \dots candidate_{k-1} \notin \mathcal{P}$ **then**
9: **continue**
10: **end if**
11: **if** $candidate \in index\_map[k]$ **then**
12: Add index j to sequence u for $candidate$
13: **else**
14: Record $(candidate, u, j)$ in $index\_map[k]$
15: **end if**
16: **end for**
17: **for all** $(candidate, u, j) \in index\_map[k]$ **do**
18: **if** $sup(candidate) < minsup$ **then**
19: Remove $candidate$ from $index\_map[k]$
20: **else**
21: $parent \leftarrow candidate_0 \dots candidate_{k-2}$
22: Remove $(parent, u, j)$ from $index\_map[k-1]$
23: Add $candidate$ to children of $parent$
24: **end if**
25: **end for**
26: k++
27: **end while**
28: $\mathcal{P} \leftarrow$ all entries in $index\_map$
29: **return** $\mathcal{P}$
30: **end function**

the starting indices of *aa* were already stored in the index map entries of *a*. Considering the earlier example of pattern *abca*, all its starting indices would already be present in the index map entries of *a*, *ab*, and *abc*.

The duplication of starting indices can be mitigated by using a tree-based approach to store the indices of the index map in a hierarchical structure:

```
{
    a: {idx: {0: [5], 1: [7]},
        children: [aa]},
    aa: {idx: {1: [6]},
        children: []}
}.
```

For each sequence motif, we store its frequent children, the starting indices of a pattern are then found under its children, keeping only indices that are unique to the sequence motif.

As the index tree structure requires less memory than the full index map, we can improve the sequence motif discovery procedure further. The candidate generation strategy from GSP can lead to the creation and checking of sequence motifs that do not appear in the database. On the other hand, PrefixSpan mines frequent sequential patterns using a depth-first search strategy, considering only candidates that are present in the sequence database using (pseudo)projection. Using the index tree entries as pseudoprojections, candidates that do not occur in D_s are not considered. This makes the sequence motif mining step more efficient. Algorithm 3 corresponds to this strategy. Note that this algorithm does not perform a Depth-First Search like PrefixSpan. Instead, sequence motifs are discovered with a Breadth-First Search where tree level corresponds to sequence length.

The first steps of the procedure are the same: a scan is made over D_s to map frequent sequence motifs of length 1 and k is initialised to 2, as we have mined the 1-patterns already (lines 2–3). Then, instead of generating candidates and sequentially recording their occurrences and checking their frequency, the index map at $k-1$ is used to generate candidates by looking at the time series directly (line 6). For every combination of $(k-1)$-*pattern parent*, time series index u, and starting index j, the item following the parent is appended to create a candidate k-*pattern* (line 7). If the other parent pattern of the candidate was not frequent (line 8), the candidate cannot be frequent either and there is no point in recording its indices. Otherwise, the indices of the candidate are recorded (line 11).

To illustrate, consider the frequent sequence motif *abc*. If its occurrence is followed by *a*, the candidate is *abca*. If we find that *bca* was not frequent, *abca* cannot be frequent either and the candidate is discarded. However, if *bca* is present in the index map, *abca* can be added as a candidate as well.

After all sequence motifs of length $k-1$ are checked, the candidates added to the sequence map are filtered on their frequency (line 14). Candidate patterns that do not comply with $minsup$ are removed (line 15). Otherwise, the indices of the candidate are removed from the parent (line 18). The candidate is then recorded as a child of the parent, to allow retrieval of all starting indices of the parent (line 19).

4.3 Removing redundant motifs

After all frequent sequence motifs have been mined using one of the methods laid out above, the less informative sequence motifs need to be removed. We consider a motif redundant if the motif is too similar to a longer frequent sequence motif. Redundant motifs are removed according to the strategy laid out in Algorithm 4.

Algorithm 4 Remove redundant sequence motifs

```
1: function REMOVE_REDUNDANT(P, omax)
2:     sort P in decreasing sm length
3:     for all sm1 ∈ P do
4:         for all sm2 ∈ P with LEN(sm2) < LEN(sm1) do
5:             if OVERLAP(sm1, sm2) > omax then
6:                 remove sm2 from P
7:             end if
8:         end for
9:     end for
10:    return P
11: end function
```

The function REMOVE_REDUNDANT receives the set of frequent sequence motifs and parameter $omax$. Line 2 sorts all sequence motifs in decreasing length. Lines 3 and 4 then loop over $\mathcal{P}$ such that shorter patterns are compared to longer patterns. Lines 5 and 6 check if the shorter sequence motif is not *redundant*, measured by OVERLAP. Redundant sequence motifs are removed. Line 10 returns the non-redundant frequent sequence motifs.

Sequence motifs are redundant if they are mostly contained in other sequence motifs. The level of overlap allowed is determined by the $omax$ parameter. Since longer motifs are more informative than shorter motifs, if two motifs greatly overlap, we keep the longer one and remove the shorter one. We define OVERLAP as

$$\text{OVERLAP}(sm_1, sm_2) = \frac{lcs(sm_1, sm_2)}{min(k_1, k_2)}, \tag{8}$$

where lcs refers to the longest common subsequence. The function removes redundant sequence motifs from the set of frequent sequence motifs and returns the non-redundant set.

The previous version of FRM-Miner also featured an $lmin$ parameter, with which users could set the minimum length for the discovered sequence motifs. As the mining algorithm generates candidates in a bottom-up manner, mining sequence motifs that are too short cannot be avoided. How-

ever, removing the patterns that are too short is a trivial step. We have therefore opted to remove the $lmin$ parameter, as a lower number of user-set parameters reduces complexity to users and, in practice, most short sequence motifs are already removed due to their overlap with longer motifs.

A more elegant and efficient solution to removing redundant patterns could be removing them during the sequence motif mining step. However, this might lead to the removal of too many patterns. To illustrate, consider the frequent sequence motifs *abc*, *abca*, and *bcaab* with $omax = 0.7$. After discovering sequence motif *abc*, we could discover *abca*. As $overlap(abc, abca) = \frac{3}{3}$, *abc* would be removed, as the overlap is higher than $omax$. We could then discover *bcaab* and as $overlap(abca, bcaab) = \frac{3}{4} > omax$, *abca* would be removed. As $overlap(abc, bcaab) = \frac{2}{3} < omax$, *abc* should not have been removed as *abca* is now absent. To prevent this removal of too many patterns, we first collect all frequent sequence motifs and then remove redundant sequence motifs from longest to shortest.

4.4 Construction of representative motifs

Algorithm 5 Map sequence motif to representative motif

1: **function** MAP($D, sm, seglen$)
2: **for all** $v, indices \in sm$ **do**
3: **for all** $(j, k) \in indices$ **do**
4: $j' \leftarrow j \times seglen$
5: $k' \leftarrow k \times seglen$
6: $cs_w^v \leftarrow t_{j'}^v, \ldots, t_{j'+k'-1}^v$
7: **end for**
8: $ao^v \leftarrow$ step-wise average of $(\{cs_r^v, \ldots, cs_s^v\})$
9: **end for**
10: $rm \leftarrow$ step-wise average of $\{z(ao^p), \ldots, z(ao^q)\}$
11: **for all** $v, indices \in sm$ **do**
12: $cs^v \leftarrow \min_{cs \in \{cs_r^v, \ldots, cs_s^v\}}(ED(z(cs), z(rm))$
13: **end for**
14: trim lengths of $\{z(cs^p) \ldots z(cs^q)\}$
15: recalculate rm
16: calculate d_{rm}
17: **return** $rm, d_{rm}, \{z(cs^p), \ldots, z(cs^q)\}$
18: **end function**

Algorithm 5 describes the MAP function, used for mapping sequence motifs to time series occurrences. This function first calculates the time series motif's start (j') and length (k') by multiplying the sequence motif's start (j) and length (k) with $seglen$ (lines 4–5). Then, the *corresponding subseries* for time series T^v are looked up in D; w identifies multiple corresponding subseries occurring in the same time series (line 6). Next, the stepwise average of the corresponding sequences in a time series, $\{cs_r^v, \ldots, cs_s^v\}$, is calculated to obtain this time series' *average occurrence* ao^v (line 8). Given the set of average occurrences in the time series, we compute a tentative *representative motif* rm as the step-wise average of z-normalised average occurrences (line 10).

The tentative representative motif is used to select the best occurrence in each time series by finding which of the occurrences in $\{cs_r^v, \ldots, cs_s^v\}$ is closest to the tentative representative motif (lines 11–13). Then, with all z-normalised best occurrences $\{z(cs^p) \ldots z(cs^q)\}$, the length is trimmed heuristically (line 14).

As sequence motifs are discovered in steps of $seglen$, the discovered length might not be optimal, as noise at the start and end of the motif might be wrongly included. To mitigate this, relatively noisy starts and ends of motif occurrences are removed, using the middle part of each motif occurrence as a reference. Concretely, the corresponding subseries are filtered to exclude the first and last symbols of the sequence motif to ensure that noise is excluded. The average difference between the stepwise maximum and stepwise minimum of the corresponding subseries is then calculated as a reference. This reference is used to select the parts at the ends of the occurrences that should fall within the motif.

After trimming the length, the representative motif is updated to the new length, using only the stepwise average of the best occurrences in each time series. Finally, the distance is calculated as defined in Sect. 2: the maximal pairwise z-normalised Euclidean distance between any two occurrences (extent) is divided by the square root of the length of the motif. Lastly, the algorithm returns the representative motif, distance, and set of corresponding subseries (line 17).

4.5 Computational complexity

The time complexity of discretisation is linear in the database size $|D|$ and sequence length $|S|$: our version of SAX has a worst case time complexity of $O(\lceil |T|/seglen \rceil \cdot \alpha)$. 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.

Sequential pattern mining has a theoretical worst-case time complexity that is exponential, i.e., given α bins there could be $O(\alpha^k)$ candidate sequential patterns up to length k. 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. That is, there are $|S| - 1$ candidate strings of length 1, $|S| - 2$ candidate strings of length 2,... and $|S| - k + 1$ candidate strings of length k 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 [6]. Assuming a maximum length $k \ll |S|$ on frequent sequence motifs, the worst-case time complexity becomes $O(|D| \cdot k \cdot |S|)$. We conclude that the proposed approach for the sequence motif discovery step is only linear in database size and the length of each sequence.

The time complexity for removing patterns that are redundant, and reducing the number 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 need to find all occurrences and then compute the distance. A naive approach to find occurrences would have a time complexity of $O(|D| \cdot |S| \cdot |\mathcal{P}|)$. However, we find 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 distance of each representative motif is quadratic in the number of occurrences, as it is the maximum pair-wise distance between occurrences, i.e., $O(|D|^2)$.

In summary, the discretisation step and the sequential pattern mining step are linear in database size. Then, a nested loop over the discovered frequent sequence motifs is performed for redundancy elimination. Lastly, the computation of representative motifs is linear in the number of remaining non-redundant frequent sequence motifs.

4.6 Space complexity

The first step of FRM-Miner is to perform the SAX transformation. This approximately reduces the size of the database by a factor of $seglen$. Though the time series database T is needed for the construction of representative motifs, a disk-aware implementation could reduce the memory requirement from $|D| \cdot |T| \cdot b_t$ to $|D| \cdot |S| \cdot b_s$, where b_t and b_s represent the number of bits used for each value in the time series database and sequence database respectively. Note that typically, $b_s \ll b_t$. Also recall that $|S| = \lceil |T|/seglen \rceil$.

The original version of FRM-Miner (Algorithm 2) performs a breadth-first search to mine sequence motifs in which for each length k, at most $((k-1) \cdot |D| \cdot |S|)^2$ candidates are generated. The index map limits the memory requirement for each pattern to the starting index of each occurrence. While infrequent sequence motifs are removed from the index map, many overlapping patterns remain. That is, the same starting index could be duplicated many times. As there are $|D| \cdot |S|^2$ frequent patterns in the worst case, the memory complexity of sequence motif mining is quadratic in the database size. Though memory complexity is normally not an issue, quadratic memory use can pose a problem; a data set of only a couple of GBs will not fit in 16GB or 32GB RAM. We therefore provide a more extensive discussion of the savings in space complexity with the improved algorithm. Our suggested improvements (Algorithm 3) limit the memory requirement during sequence motif mining, as the indices of children of frequent sequence motifs can be removed from their parents. For any two sequence motifs of different length with the same starting index, one must be the parent of the other. Thus, each index can only occur once in the final index map. Concretely, the initial scan of the database that discovers sequence motifs of length 1 will cause every index in the database to be recorded once, which has size $|D| \cdot |S|$. For subsequent lengths k, the indices of possible sequence motifs are added to the index map. As the proposed new version of FRM-Miner does not perform a candidate generation step after which all candidates are checked subsequently, the index map needs to keep track of all sequence motifs of length k it is currently evaluating. In the worst case, every index is the start of a frequent sequence motif with a length below k and all candidate sequence motifs are added as well, resulting in a total size of $2 \cdot |D| \cdot |S|/k$, which is linear in database size.

Even though we observe a reduction in the number of sequence motifs from the redundancy elimination step in practice, redundancy elimination is skipped if $omax = 1$. In the typical case of $omax < 1$, the only sequence motifs that are removed in the worst case are parent of other sequence motifs. The redundancy elimination step thus can reduce the number of sequence motifs, though the number of indices recorded is not affected in the worst case. This step thus has no effect on the worst-case space complexity of FRM-Miner.

The last step is the construction of representative motifs. This step expands the data stored for each sequence motif: instead of only the starting indices of the occurrences, each value of the corresponding subseries is stored, as well as the average occurrence and representative motif. This leads to a worst-case space complexity that is linear in the length of each respective sequence motif.

To summarise, the discretisation step can greatly reduce the memory requirements of FRM-Miner. The new version of FRM-Miner can reduce the memory complexity for the discovery of sequence motifs from quadratic to linear. The last step of FRM-Miner, which constructs the representative motifs from the occurrences, can increase the required memory, though only linearly.

4.7 Improving precision

As already pointed out in Sect. 4.1, we apply global z-normalization to each time series instead of normalizing subsequences. The implication of this is that observations outside of a motif influence its symbolic representation. We point out that the same is true for algorithms based on the Matrix Profile in all but the cases where the motif length is known in advance. Given this, we still opt for global normalization because we discover motifs of varying length and do not assume that the correct length is known in advance.

A side effect of global z-normalization is that time series databases with concept drift can contain frequent motifs where subsequence normalization would result in a small distance but the SAX representation gives vastly different

symbols. Luckily, stationarity in time series can easily be assessed visually or with statistical tests. A non-constant mean can be mitigated by differencing each time series in the database δ times before normalization. Non-stationarity in the variance can be mitigated with other techniques, such as the Box-Cox transformation [5].

Discretisation with the SAX representation makes FRM-Miner an approximate algorithm via two other mechanisms: the PAA step can divide motif occurrences into different segments, causing noise to be included in the start and end of the symbolic representation. This can in turn change the first and last symbols of the discrete representation, causing occurrences to be missed. The other mechanism is because of the discretisation step: two segments with arbitrarily small distance can be binned into neighbouring symbols. Again, this can cause occurrences to be missed, because they vary in their symbolic representation. Mitigating the effect of discretisation would require us to consider neighbouring symbols to match. That is, the sequence *ab* would need to be considered a match with *bb*, *ba*, *bc*, *aa*, and *ac*. It is easy to see that for longer patterns, the number of matches to consider would grow exponentially and thus explode the complexity of sequence motif discovery. Subsequently, the number of occurrences for each sequence motif would grow exponentially, also increasing the time required for mapping sequence motifs to time series motifs.

As stated above, the variability in discrete representations might cause FRM-Miner to miss occurrences. Missing enough occurrences to make the motif too infrequent to be discovered would lead to a false negative regarding our problem statement. Missing some occurrences but not enough to bring support below $minsup$ can be solved by applying MASS [46] to the time series in which no occurrence of the motif has been found. Slightly lowering $minsup$ can reduce the number of false negatives at the cost of more false positives, which can be filtered out in post-processing.

Along with false negatives due to similar segments having different representations, false positives can arise due to differing segments having the same representation. Though the SAX representation provides a lower bound on the distance between segments, the minimal distance between two neighbouring symbols is 0. The reason for this is, as we mentioned earlier, that two very similar segments can still have differing representations. Therefore, a discovered sequence motif may not result in a set of similar subsequences.

The risk of dissimilar segments being represented with the same symbol increases with the compression rate. That is, higher values for $seglen$ and, particularly, low values for α increase the risk of finding false positives.

This effect is mitigated by our last step of ranking the discovered motifs on their distance. As the distance of a spurious motif is much higher than that of a true motif, false positives naturally have a low impact. A visual inspection of the top ranked motifs can help determine if the motifs are spurious or meaningful and whether they contain (many) false positive matches. If the motifs discovered with a certain parameter combination are found to be uninformative, the parameters can be adjusted to see if other settings result in better motifs. When using the motifs for downstream tasks, a quality filter that removes motifs with very large distances could be considered.

5 Experiments

After introducing the experimental set-up of the experiments, we start experimentation with illustrating our intended problem setting with simulated data and demonstrating real-world applicability of FRM-Miner in three case studies. We continue the experiments with a comparison of FRM-Miner 1.0 and FRM-Miner 2.0 in terms of run time and peak memory use.

Then, we compare the ability of FRM-Miner to find motifs in noisy settings with the performance of Ostinato. 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).

The desired motif length is Ostinato's sole parameter. To create the most beneficial conditions for Ostinato, we provide it with the ground-truth length.

We further study FRM-Miner's scalability by measuring the impact of database size and time series length on run time and peak memory use. Subsequently, we assess the impact of key parameters on run time and number of discovered motifs.

5.1 Experimental set-up

In order to experimentally evaluate FRM-Miner, we provide a Python implementation that is compatible with Python 3.9 and later versions.

In our prior work, we provided two implementations: a Python-interfaced C++ implementation and a pure Python implementation. Where applicable, we refer to the Python implementation from the prior work as FRM-Miner 1.0 and refer to the implementation of the improved algorithm as FRM-Miner 2.0. If we do not specify the version, we are referring to FRM-Miner 2.0.

In experiments that compare against Ostinato, we use implementation that is provided in the Stumpy library [20]. All experiments are run with Python 3.12.1 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.

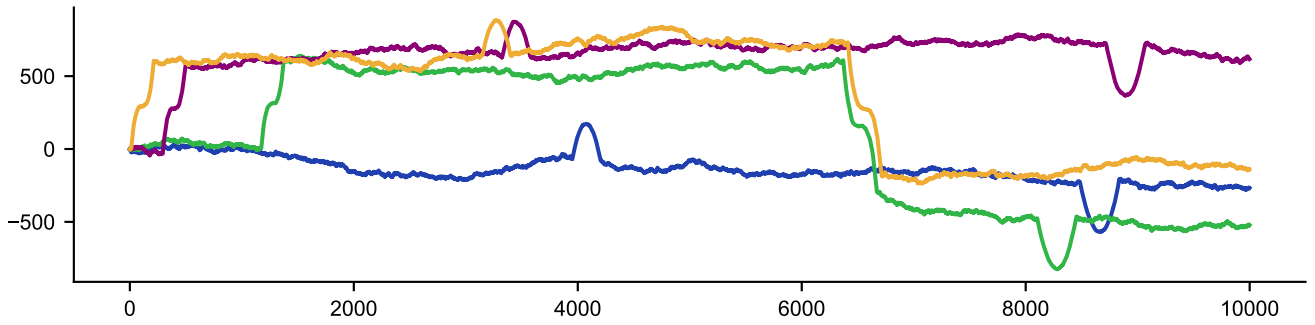

Fig. 4 A selection of 4 out of 10 000 random walk time series with embedded motifs. Up to four motifs are inserted into each random walk in the database

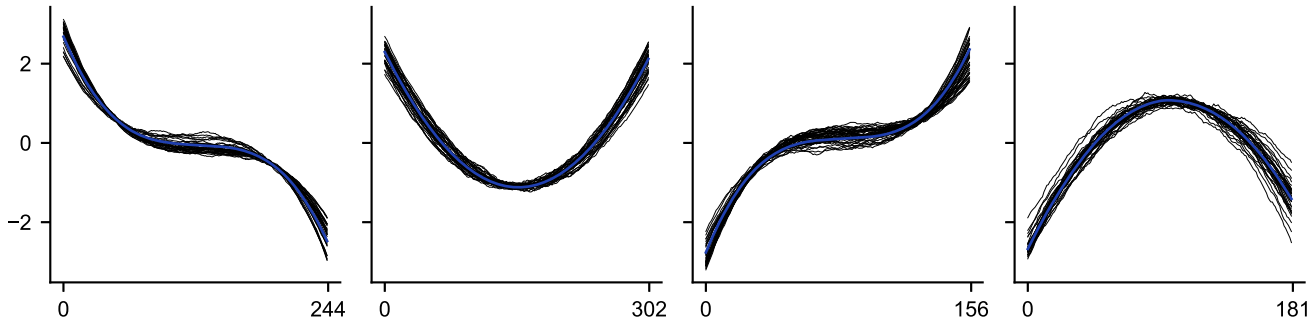

Fig. 5 Top 4 motifs discovered with FRM-Miner. Parameter settings are $minsup = 0.3$, $seglen = 30$, $\alpha = 4$, $omax = 0.6$, $\delta = 1$

5.2 Problem setting

To illustrate our problem setting, we create a time series database containing random walks and embed motifs in the random walks. We start with generating 10 000 time series of length 10 000. Each time series is generated as 10 000 observations of Gaussian noise. We insert four distinct motifs with varying length and varying support in the database. The motifs vary in length from 200 to 350 in steps of 50. The support of the motifs varies from 90% to 75% in steps of 5 percentage points. For each motif, starting indices are sampled in different 'regions' of the Gaussian noise to prevent overlap between motif occurrences. Then, the stepwise cumulative sum of each time series is calculated to transform the time series into random walks with embedded motifs. A selection of four time series is shown in Fig. 4. Because existing algorithms require that motifs occur in every time series in the collection and/or require a single predefined length, existing approaches are unfit for this problem setting.

As becomes clear from the figure, the presence of the first and third motifs causes subsequent motif occurrences within a time series to occur on vastly different scales. However, we mitigate the differing scale of occurrences by setting $\delta = 1$, discovering frequent motifs in the first derivative of the series. The resulting representative motifs and their occurrences are shown in Fig. 5.

Table 1 Aggregate results for 10 000 simulations

Motif	Discovery (sd)	Recall (sd)	Length (sd)
Up	99.9% (0.03)	42.5% (0.01)	76.8% (0.03)
Peak	100.0% (0.00)	38.3% (0.01)	73.6% (0.01)
Down	100.0% (0.00)	39.5% (0.01)	80.9% (0.01)
Dip	100.0% (0.00)	46.0% (0.01)	86.2% (0.00)

Clearly, the four motifs are retrieved. Note that for none of the motifs, the entire length is covered, neither are all occurrences of the motifs discovered. We perform the experiment a total of 10 000 times and measure for each run whether the motifs are discovered, the recall, and the length recovery. We count a motif as discovered if at least half of the occurrences of a representative motif overlap with at least half of the ground truth locations of an embedded motif. The recall measures the fraction of embedded occurrences discovered by a motif. The length recovery is calculated by dividing the discovered length by the ground truth length. The averaged results per motif and their standard deviations are shown in Table 1. The table shows the motifs in the order that we embed them in the time series, as can be seen in Fig. 4. We note that this is not necessarily the order in which the discovered motifs are ordered. For instance, Fig. 5 ranks the motif 'Down' first, followed by 'Dip', 'Up' and 'Peak'. This order differs across runs.

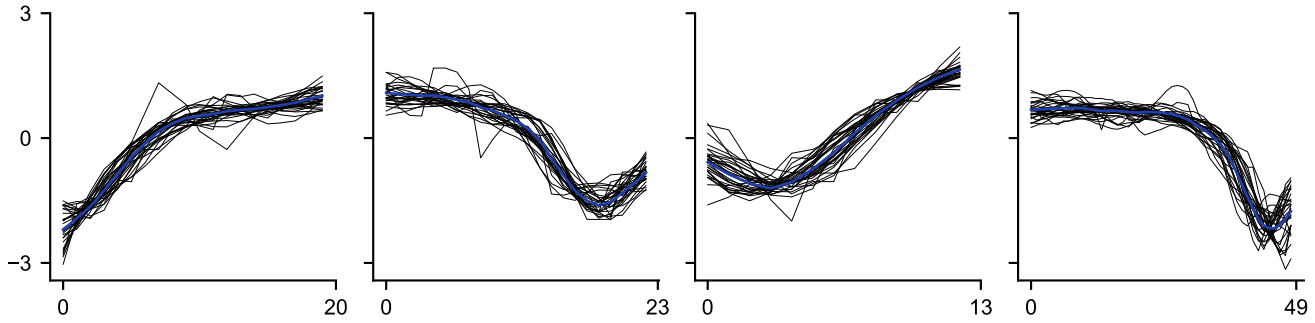

Fig. 6 Frequent motifs of variable length in cycling speed data. Parameter settings are $minsup = 0.3$, $seglen = 6$, $\alpha = 4$, $omax = 0.8$ (color figure online)

From the table, it becomes clear that the recall is low for all ground truth motifs. The behaviour that we observe here is typical for what we have observed in other experiments: most non-redundant sequence motifs have a support just above the $minsup$ threshold. Sequence motifs with higher support keep growing until the support decreases below the threshold.

We argue that the low recall does not pose a problem for FRM-Miner. Our goal is to discover motifs that occur frequently. Though not all occurrences are retrieved, the discovery rate is perfect for three of the motifs and almost perfect for the other motif. In cases where more exact discovery of occurrences is necessary, the support threshold can be increased. Increasing the support threshold comes with the cost of reducing the length retrieval rate. The reason behind this is that sequence motifs are grown until they are not frequent any more. By increasing $minsup$, this threshold would be encountered earlier in the process.

Another way of increasing the recall is described in Sect. 4.7, where we propose to query time series without a match for the discovered representative motif using MASS [46]. This post-processing step would search in each time series without an occurrence of the motif for the best matching subsequence. Additional subsequences that do not increase the motif's distance could be included. Alternatively, a factor η by which the distance is allowed to grow as a result of including more matches could be used as well. We note that this increases risk of introducing some false positive matches.

In conclusion, we stress that the results of FRM-Miner are *approximate*, but nevertheless very good, in a problem setting that existing exact algorithms cannot handle.

5.3 Cycling speeds

To illustrate the real world applicability of FRM-Miner, we mine frequent motifs on speed data collected in 93 distinct cycling training sessions of one athlete with a GPS bike computer. The bike computer records various features, such as latitude and longitude, speed and altitude, at each second. The shortest training session was 680 s (11.33 min) and the longest was 30 629 seconds (8.51 h). The database contains 637 304 observations in total.

After anonymizing the recordings by removing latitude and longitude information, we extract the speed information from each second in the recording to create a time series database of cycling speeds. We apply FRM-Miner to this time series database.

We set $minsup = 0.3$, which is used throughout the rest of the experiments. The $seglen = 6$ corresponds to a tenth of a minute per segment. The choice of setting $\alpha = 4$ keeps a balance between the noise reduction provided by SAX and the ability to distinguish different values. Lastly, we recommend $omax = 0.8$ as a reasonable starting point for most analyses.

Figure 6 shows the top 4 patterns discovered from the cyclist's data, with individual occurrences shown in black and the mean pattern in blue. From the figure, some patterns in the cycling behaviour of the athlete can be distinguished. Motif 1 (20 s) captures a moderate acceleration from well below average speed to a plateau just above average. Motif 2 (23 s) begins with a relatively stable period slightly above the average speed before descending steeply to well below average and a small recovery of speed at the end. This motif could represent a consistent deceleration event such as braking or a short uphill effort. Motif 3 (13 s) is the shortest and most intense, showing a sharp and continuous rise from below to well above average speed. This represents a brief explosive acceleration. Motif 4 (49 s) is the longest motif. It begins near average speed and then declines steadily back below average. This motif represents a sustained intensity followed by braking quickly. Note that the motifs have various lengths (though not necessarily so), and that the optimal length of the motifs is not a multiple of $seglen$, as a result of trimming the noise as described in Sect. 4.

5.4 Insect telemetry

Next, let us consider the time series database shown in Fig. 7. This database contains four recordings of Asian citrus psyllid (*Diaphorina citri*) behaviour made with an electrical

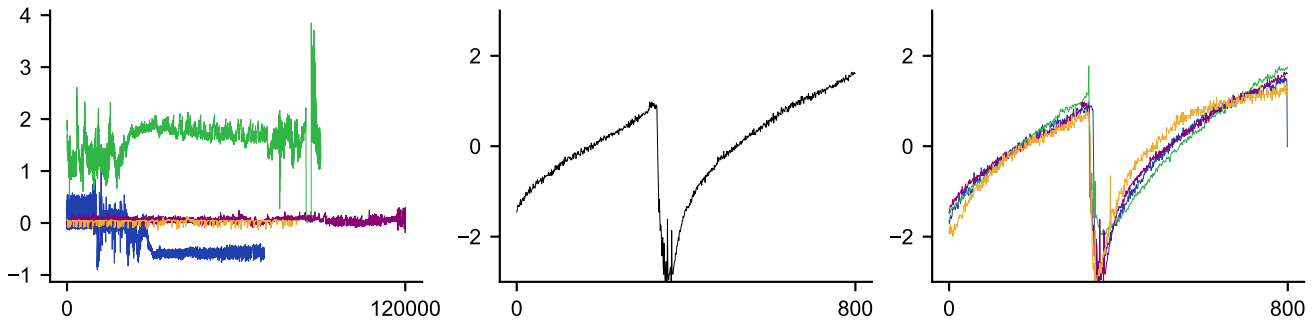

Fig. 7 Four EPG telemetry recordings of Asian citrus psyllid behaviour (left). Data set taken from Kamgar et al. [17]. The z-normalised consensus motif of length 800 (middle). Motif set of consensus motif and nearest neighbours in the other three time series (right)

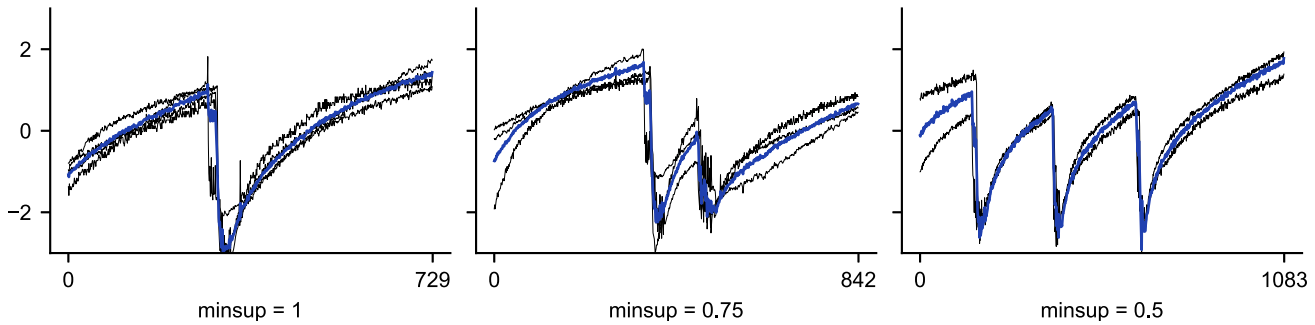

Fig. 8 Representative motifs and corresponding motif sets discovered in insect telemetry data. Parameter settings for $minsup$ are shown on the x-axis labels. The other parameter settings are $seglen = 30$, $\alpha = 4$, $omax = 0.8$, $\delta = 1$. The left representative motif corresponds to the motif found by Ostinato

penetration graph (EPG) and was originally presented by Kamgar et al. [17]. As explained in this work, conserved patterns in the insects' EPG recordings are useful in understanding and controlling their behaviour.

The authors present a well-conserved consensus motif of length 800 that can be discovered with Ostinato [17]. 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.

From the figure, it becomes apparent that this time series database is particularly noisy. After each time series is z-normalised, the consensus motif still occurs at different levels, because of the non-stationarity of some of the time series in the database. To deal with the non-stationarity, we apply FRM-Miner on the first difference of the time series database, setting $\delta = 1$. The parameter settings for α and $omax$ are identical to those of the previous experiments. Because the time series in this experiment are collected with a period of 100Hz as opposed to one recording each second in the previous experiments, we increase the value of $seglen$, setting it to 30.

Figure 8 shows the result of applying the indices of the discovered motifs to the original time series. The motif that results from setting $minsup = 1$ closely resembles the consensus motif found by Ostinato. Note that FRM-Miner automatically determined that the optimal length of this motif is 729, rather than 800 that was given to Ostinato [17]. Looking at the three occurrences plotted in Fig. 7 (right), we can see quite some divergence at the start and the end of the motif of length 800, which is why FRM-Motif decided to trim those parts and settle on a length of 729.

If we relax the $minsup$ from 1 to 0.75 or 0.5, we find two other well-conserved patterns that occur in the time series. These patterns seem related to the same "phloem salivation" behaviour as the consensus motif [17]. These motifs were previously unknown and could not be discovered with Ostinato, as they do not occur in each of the four time series.

5.5 Eye tracking

For a third case study on real world data, we turn to eye movement data collected in the context of a study on expertise [16]. The goal of the study was to find differences in eye movement behaviour across different levels of expertise in screen-based tasks. In this study the task was playing Tetris. Participants were invited to partake in 40-minute sessions that included completing a survey with demographic data, an introduction to the game controls, a short practice round and the study itself. In the study, participants played Tetris

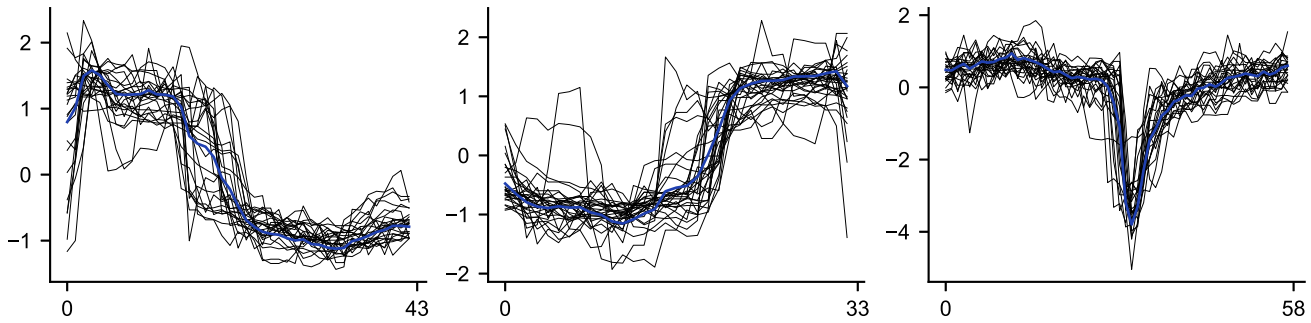

Fig. 9 Representative motifs and corresponding motif sets discovered in eye movement data. The first two motifs were discovered in the *gaze_angle_y* signal and correspond to looking up and down. The third motif was discovered in the *EAR* signal and corresponds to a blink. Parameter settings for both signals were set to the same values of $minsup = 0.3$, $seglen = 10$, $\alpha = 4$, $omax = 0.9$, $\delta = 1$

for 13 min while their faces were recorded with Open Broadcaster Software at 30 frames per second (fps).

The videos were subsequently processed to extract time series signals. With the *cvzone* Python library, the eye aspect ratio (*EAR*; distance between eyelids) was extracted. Then, OpenFace [4] was used to extract the *gaze_angle_x* and *gaze_angle_y* signals. As there are 13 min of data per participant recorded at 30 fps for 79 participants, there are 23 400 observations in each signal per participant and 1 848 600 observations for each signal in total.

Prior research on expertise classification with these signals found that the *gaze_angle_y* and *EAR* signals are most important in this data set, while *gaze_angle_x* is less important [32]. This is not surprising, as Tetris is a vertically oriented game. We apply FRM-Miner to the *gaze_angle_y* and *EAR* signals. The results are shown in Fig. 9. The first two motifs in the figure are discovered in the *gaze_angle_y* signal, while the third motif is found in the *EAR* signal.

In the figure, we can see behaviour in the *gaze_angle_y* signal that is consistent with looking up and down. This behaviour is very typical to the game of Tetris, as shapes appear at the top of the screen and need to be placed at the bottom. The motif discovered in the *EAR* signal is consistent with the well-known pattern of blinking, as the distance between the eyelids rapidly decreases during a blink. The patterns we discover are thus meaningful and consistent with prior domain knowledge.

5.6 Run time comparison of FRM-Miner 1.0 and 2.0

In order to measure the difference in run times between FRM-Miner 1.0 and FRM-Miner 2.0, we apply both versions of the algorithm with the same hyperparameter settings to the data sets in the UCR Time Series Archive [8, 9]. We choose this archive because of the varying sizes and ratios between numbers of time series and time series lengths.

As the intended purpose of this archive is to compare performance of time series classifiers, the data sets are supplied as separate train and test sets. We join the train and test sets into a single time series database and apply both versions of FRM-Miner, recording the total time to find and rank the frequent representative motifs.

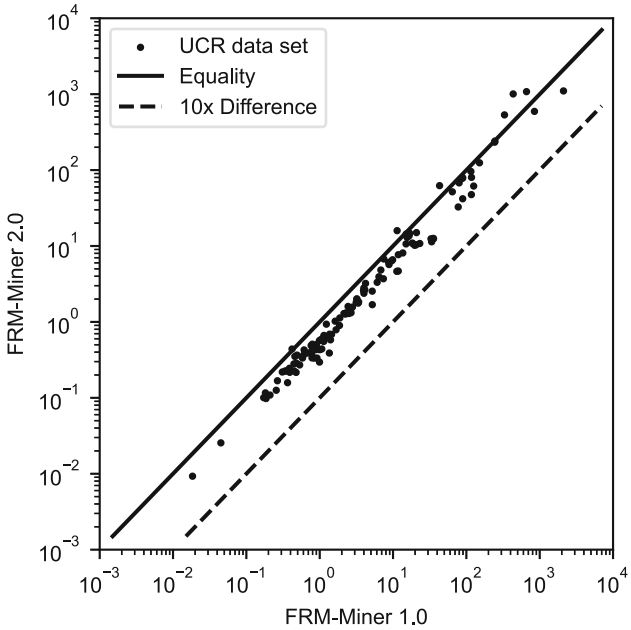

Fig. 10 Run time comparison of FRM-Miner 1.0 and FRM-Miner 2.0. Each dot represents one data set in the UCR Time Series Archive. Run times are measured in seconds. Both axes use a base 10 log scale. The total run times are 1.87 h for FRM-Miner 1.0 and 1.68 h for FRM-Miner 2.0 with the parameter settings $minsup = 0.3$, $seglen = 2$, $\alpha = 4$, and $omax = 0.8$

Though the optimal parameter settings are data dependent and can vary across data sets, we apply the same parameters to each data set. The settings are $minsup = 0.3$, $seglen = 2$, $\alpha = 4$, and $omax = 0.8$. We set $minsup$, α, and $omax$ to the same values used throughout the previous experiments. We set $seglen = 2$ as this is the minimal length of an individual time series in the archive.

The results, shown in Fig. 10, indicate that FRM-Miner 2.0 is typically a bit faster than FRM-Miner 1.0. This is also

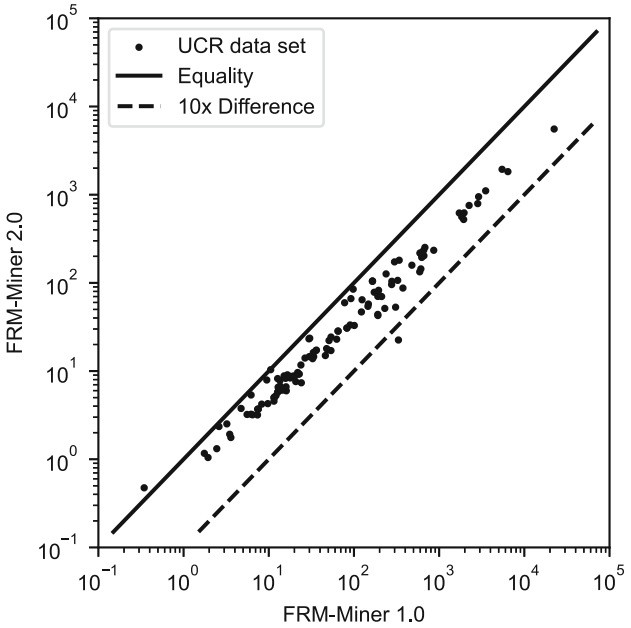

Fig. 11 Peak memory use comparison of FRM-Miner 1.0 and FRM-Miner 2.0. Each dot represents one data set in the UCR Time Series Archive. Peak memory use is measured in megabytes. Both axes use a base 10 log scale. The parameters settings are $minsup = 0.3$, $seglen = 2$, $\alpha = 4$, and $omax = 0.8$

reflected in the total run time: FRM-Miner 1.0 took 1.87 h to process the entire archive, while FRM-Miner 2.0 took 1.68 h. Thus, even though FRM-Miner 2.0 is faster than FRM-Miner 1.0, the difference is small.

Looking more closely at the results, the runtime improvement is quite consistent: FRM-Miner 2.0 outperforms FRM-Miner 1.0 on 122 out of 128 data sets. The six data sets where FRM-Miner 2.0 is slower are found in the upper-right region of the figure, which indicates that they are among the harder, longer-running data sets.

Recall that FRM-Miner 1.0 generates candidates by combining frequent sequence motifs of previous lengths. This might generate many redundant candidates that do not occur in the sequence database. FRM-Miner 2.0, on the other hand, generates candidates by checking which symbols follow a frequent sequence motif. We hypothesize that the longer running data sets contain a higher variety of frequent sequence motifs. In that case, a relatively small fraction of candidates generated by FRM-Miner 1.0 is redundant and the checks by FRM-Miner 2.0 introduce overhead that turns out to be unnecessary.

5.7 Peak memory comparison of FRM-Miner 1.0 and 2.0

While the speed-up of FRM-Miner 2.0 over FRM-Miner 1.0 is not very large, the motivation for most changes to the algorithm was to increase memory efficiency. Therefore, we compare the two in terms of peak memory use during the processing of the UCR Time Series Archive. The outcomes are shown in Fig. 11

Because peak memory usage defines the amount of available memory necessary to be able to run the algorithm, we consider it the most important outcome to measure memory performance. We remark that the measurement of memory usage severely impacts run times, especially on larger data sets. Therefore, we carry out the run time and memory experiments separately. The parameter settings are kept the same.

It becomes apparent from Fig. 11 that the improvement of FRM-Miner 2.0 over FRM-Miner 1.0 increases with data size. The data set where both versions have the smallest peak in memory use is SmoothSubspace. This is the only data set where FRM-Miner 1.0 outperforms FRM-Miner 2.0, peaking at about 0.35MB and 0.47MB respectively. All other data sets have a lower peak in memory use when FRM-Miner 2.0 is applied to it than when applying FRM-Miner 1.0. Moreover, the variability in memory use is much lower for the improved version. The peak memory use of FRM-Miner 1.0 varies from around 0.34MB to around 22.38GB. In contrast, the peak memory use of FRM-Miner 2.0 varies from just around 0.47MB to around 5.55GB. The difference in memory use increasing for larger data sizes is in line with a quadratic memory complexity for FRM-Miner 1.0 and linear memory complexity for FRM-Miner 2.0.

5.8 Robustness to noise

Continuing our experimentation, we ask *to what extent is FRM-Miner able to find motifs in noisy settings?* To explore this, we create a database of 100 random walk time series of length 10 000. A constructed motif, consisting of a length 512 sinusoid pattern, is inserted into 75 randomly selected time series at random locations. We add increasing levels of Gaussian noise to the motif occurrences, varying the standard deviation between 0.4, 0.6, 0.8, 1.0 and 1.2. Figure 12 shows three of the resulting time series with noise level 0.6. While all three of the time series contain a motif in different locations, only 75 of the 100 time series in the database contain an embedded motif.

From the figure, it becomes apparent that the random walk nature of the time series causes the motif occurrences to occur both at varying levels and with varying magnitudes. As we are interested in similarity after z-normalisation of occurrences, this should not matter for motif discovery.

We apply FRM-Miner with the hyperparameter settings $minsup = 0.3$, $seglen = 25$, $\alpha = 4$, $omax = 0.8$ and $\delta = 1$. The resulting top motif sets for each noise level are shown in Fig. 13. The figure shows that the motif is retrieved for all noise levels, though not at the correct lengths. The length of the top motif is decreasing in the amount of noise

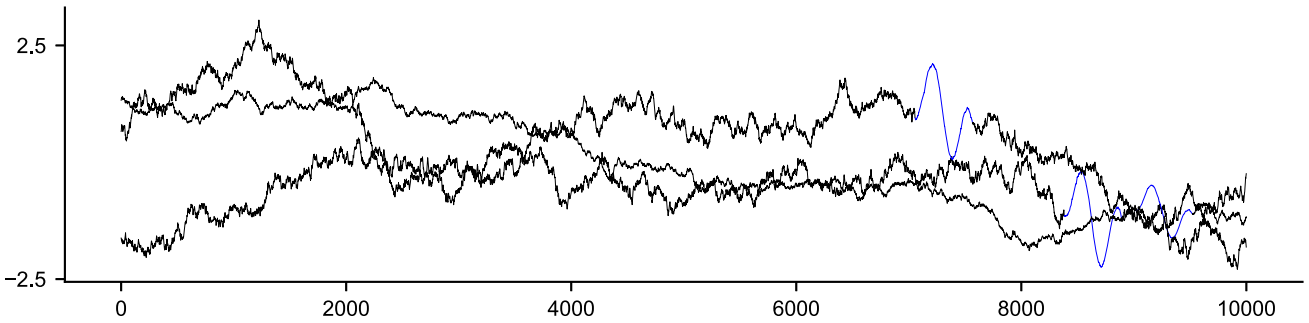

Fig. 12 Simulated random walk time series database with embedded motifs. Motif occurrences are shown in blue (color figure online)

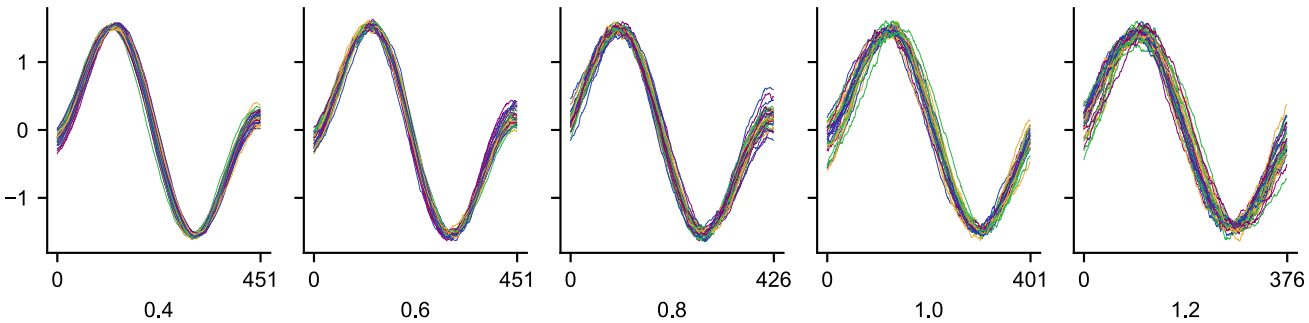

Fig. 13 Top motifs discovered by FRM-Miner for the 5 noise levels shown on the x-axis labels. Motif injected into 75% of time series. Each column corresponds to the level of noise in the axis label and contains the top motif for that noise level

added to motif occurrences. However, the highest noise level still results in 73% (376/512) of the correct motif length.

We compare FRM-Miner to Ostinato, carrying out the same experiment, but giving Ostinato the correct motif length in advance. As shown in Fig. 14, Ostinato performs much worse in retrieving the motif than FRM-Miner, even when given the correct motif length. For all noise levels, the consensus motifs is a part of the random walk without overlapping the injected motifs. Note, too, that Ostinato returns a particular individual occurrence of the discovered motif, while FRM-Miner finds the motif sets directly, without requiring an additional step to find all matches to the consensus motif. Of course, had the motif been injected into all of the time series, Ostinato would be able to recover the motif if given the correct motif length. However, the situation where a motif occurs frequently but not in all time series is more realistic.

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

5.9 Scalability of FRM-Miner

Next we ask *how does data size affect the run times and peak memory use of 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 100 to 10 000. Each combination of time series quantity and length is evaluated 10 times to mitigate randomness in system noise.

For each of the three possible values of time series quantity, we average the run times of all settings that correspond to that value. That is, to obtain the run time for a certain time series quantity, the run times of time series length 100, 1 000, and 10 000 with that time series quantity are averaged. Similarly, the run time values for time series length are obtained by averaging the run times for each time series quantity. We obtain the results for all combinations that lead to a certain total database size by averaging them as well. Note that different total database sizes can have different numbers of combinations that lead to these sizes. For instance, the sizes 10^4 and 10^8 can only be obtained with a single combination. Contrarily, the size 10^6 can be obtained in 4 ways.

To validate our findings, we compare them with the run times from Sect. 5.6. For each data set in the UCR Time Series Archive, we record the time series quantity, average length, and total database size, plotting these values against the run times.

The results, depicted in Fig. 15, differ for the simulation and the UCR Time Series Archive. However, both reveal an approximately linear relationship between data size and

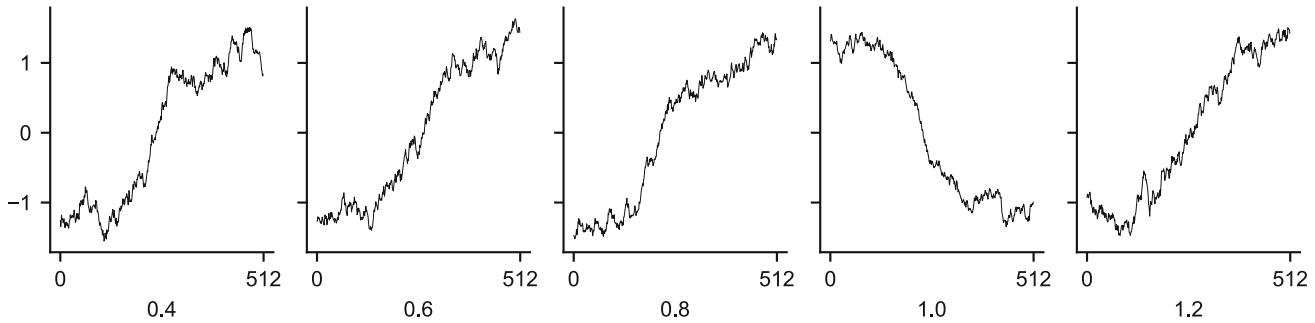

Fig. 14 Consensus motifs found by Ostinato for the 5 noise levels shown on the x-axis labels. Motif injected into 75% of time series. The consensus motifs do not correspond to the injected motif and contain parts of the random walks instead

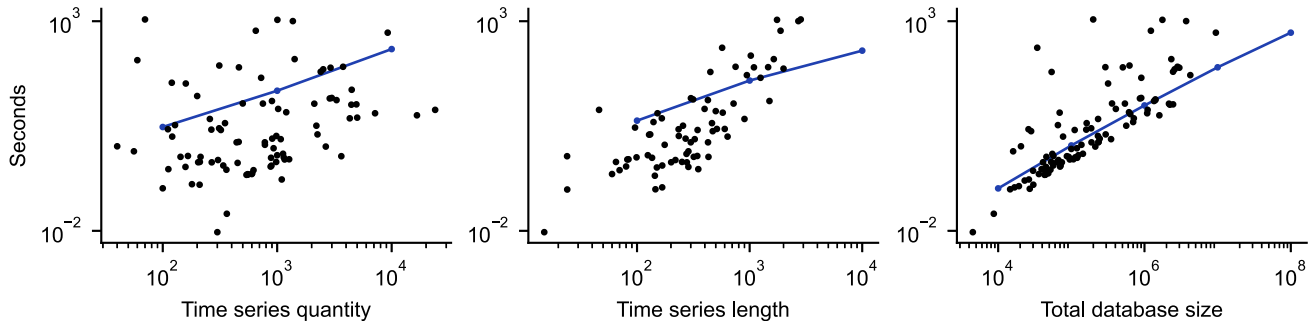

Fig. 15 Run times for different simulated data sizes (blue) and data sets in the UCR Time Series Archive (black). We observe an approximately linear relationship for time series quantity, length, and total database size, measured as number of data points (color figure online)

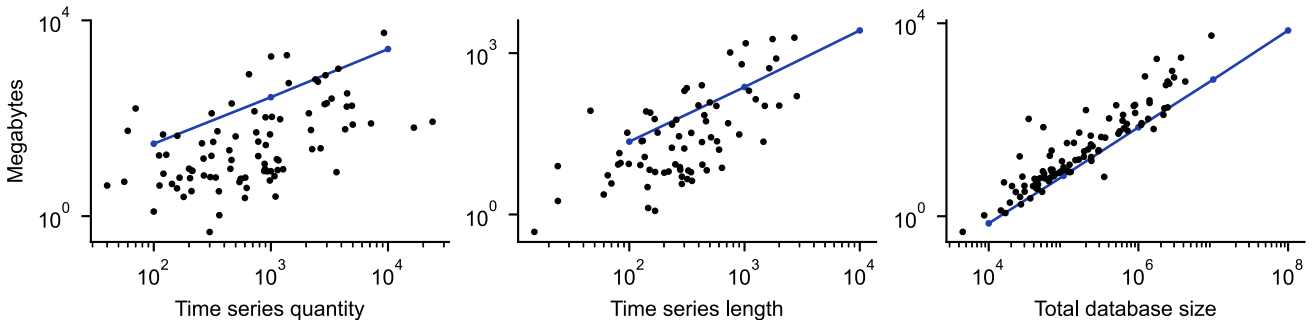

Fig. 16 Peak memory use for different simulated data sizes (blue) and data sets in the UCR Time Series Archive (black). We observe an approximately linear relationship for time series quantity, length, and total database size, measured as number of data points (color figure online)

run time. For time series quantity and time series length, the linear upwards trend is very clear for the simulation, whereas the UCR Time Series Archive shows a weaker relationship, where a small elbow is visible as well. As the time series quantity contains more noise for the UCR Time Series Archive as well, we regard the non-linearity in this trend as random variation. This difference can be explained by the time series quantity and time series length comparisons being ceteris paribus in the simulation. That is, for two different values of time series quantity, the exact same values for time series length are applied. This is not the case for the UCR Time Series Archive, as there likely is a relationship between time series quantity and length. This explanation is supported by the reduced variability in the UCR Time Series Archive results and the congruent trends in the total database size.

The run times on the UCR Time Series Archive for total database size are slightly higher than on the simulation. The most likely explanation for this is that real-world data contains more naturally occurring patterns than the injected simulated motifs. However, both experiments show a similar trend. We conclude that both time series quantity and time

Table 2 Parameter settings for parameter impact experiment

Parameter	Settings
$minsup$	0.1, 0.3, 0.5, 0.7, 0.9
$seglen$	10, 20, 30, 40, 50
α	2, 3, 4, 5, 6, 7, 8, 9, 10
$omax$	0.5, 0.6, 0.7, 0.8, 0.9

series length affect the run time of FRM-Miner, but only to the extent that they increase the size of the total database.

We repeat this experiment, measuring peak memory use for different data sizes. As before, we add the results obtained on the UCR Time Series Archive to show scalability on real-world data. The results, shown in Fig. 16, look very similar to those in the run time experiments. Again, we find linear trends in increasing time series quantity and length, as well as total database size. As in the run time experiment, we find that the variability in the results is smaller for the total database size than for the time series quantity and average time series length.

5.10 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 larger data sets, we collect 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 apply FRM-Miner to this collection of time series, varying the values of the four parameters $minsup$, $seglen$, α, and $omax$. The range of values for each parameter is shown in Table 2.

Figure 17 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 three parameters.

From the figure, we can see that the relationships between run times and number of discovered patterns are similar for $minsup$, $seglen$, and $omax$. Unsurprisingly, $minsup$ and $seglen$ have a negative relationship with both outcomes, as higher values for $minsup$ reduce the number of frequent sequence motifs and higher values for $seglen$ shrink the size of the sequence database. In contrast, increasing $omax$ decreases the number of redundant patterns, so more representative motifs have to be constructed.

The run times decrease with α, while the number of patterns discovered increases. For smaller alphabets, more segments are assigned to the same discrete value, so more patterns are considered frequent and the sequence motif mining step will take longer. However, for these lower α values, there is much more overlap between the discovered sequence motifs, so relatively more motifs will be considered redundant. On the other hand, for higher values of α, there is a larger number of possible sequence motifs, as more combinations can be made with the alphabet. Because this will lower the amount of frequent sequence motifs, the time required for the sequence motif mining step is reduced. However, less of the discovered sequence motifs will be redundant, which results in a larger number of discovered patterns.

5.11 Parameter settings

FRM-Miner has five user-defined parameters: $minsup$, $seglen$, α, $omax$ and δ. Their settings are adaptable based on the data characteristics. Because of the 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 observe that lower values increase precision, whereas higher values decrease run times. We recommend starting with relatively higher values for faster experimentation and lowering $seglen$ if additional precision is necessary. The optimal value for α is dependent on the data set at hand. However, throughout all experiments, $\alpha = 4$ provided adequate results. Beyond these parameters, fine-tuning is available using $omax$. We propose $omax = 0.8$ and use it exclusively in the case studies. However, if the discovered motifs are found too similar, lower values can be considered. Higher values will increase both precision and the number of discovered motifs, at the risk of most extra motifs being similar in shape to motifs that have already been found. Lastly, the parameter δ can be used to increase precision if a substantial number of time series in the collection contain concept drift. In this case, setting $\delta = 1$ can help to make the time series stationary.

With regards to parameter experimentation, we furthermore remark that, while sequence motifs discovered with different parameter combinations might not be comparable, time series motifs discovered with these settings are comparable. That is, the outputs of several runs of FRM-Miner are comparable and can be ranked to find the setting leading to the best frequent representative motifs or frequent motif sets. Furthermore, outputs of different runs could be combined and ranked as well. Note, however, that combining the outputs of several runs of FRM-Miner with different settings may result in unintended redundancy, since motifs deemed redundant in one run may yet find their way into the output in another run.

6 Discussion and conclusion

This work proposes FRM-Miner, an algorithm for the discovery of frequent representative motifs in large collections

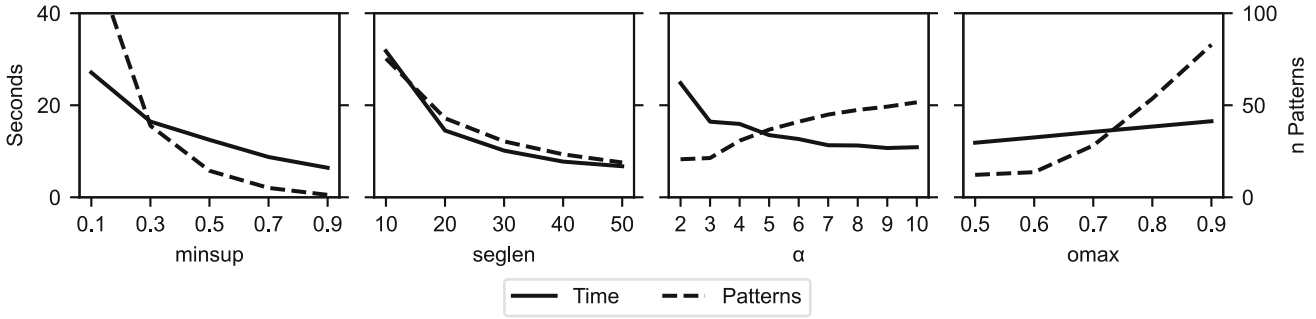

Fig. 17 Average run time in seconds and number of discovered motifs for different values of *minsup*, *seglen*, α, and *omax*

of time series. With sequential pattern mining methodology, we efficiently discover motifs, determining each motif's optimal length automatically. The discovery of sequence motifs is followed by a redundancy elimination step to increase the diversity in the set of discovered top motifs. The time series occurrences of non-redundant sequence motifs combined to form motif sets and a representative motif. Subsequently, the discovered motifs are ranked based on their ability to generalise occurrences.

While the discrete SAX representation gives FRM-Miner the ability to discover frequent motifs very efficiently, it could lead to a loss in accuracy through segmentation and binning. This drawback can be mitigated by experimenting with different parameter settings (e.g., smaller segments or more bins). Notably, the efficiency of FRM-Miner allows for fast experimentation with different parameter settings. The ability to quickly experiment with different parameter settings also reduces the drawback of having several parameters to tune.

If finding the exact number of matches of a motif in a time series database is a priority, the output of FRM-Miner can be used as seed sequences for further analysis, for instance using MASS [46]. Directions of future work include exploring the usefulness of the retrieved motifs for tasks such as clustering, classification and anomaly detection, as well as motif discovery in multivariate data sets. Another venue for future work could be the use of novel discretisation techniques [3].

In conclusion, FRM-Miner enables mining motifs of varying length on large collections of time series. This is an important innovation, as motif length and support are typically hard to know in advance. We propose improvements over an earlier version of FRM-Miner that greatly reduce memory use and provide a small run time improvement as well. We extend our prior work by allowing motifs of any length, fine-tuning lengths of the discovered motifs. Our experimental evaluation shows FRM-Miner's scalability, precision, and real-world applicability.

Author Contributions All authors contributed to the design of the work, the writing and review of the manuscript, and design, analysis and interpretation of experiments.

Data Availability Access to the data is provided within the manuscript.

Declarations

Conflict of interest The authors declare no conflict of interest.

References

1. Aalbers, G., Hendrickson, A.T., Vanden Abeele, M.M., Keijsers, L.: Smartphone-tracked digital markers of momentary subjective stress in college students: idiographic machine learning analysis. JMIR Mhealth Uhealth **11**, e37469 (2023)
2. Andrysiak, T., Saganowski, Ł, Kiedrowski, P.: Anomaly detection in smart metering infrastructure with the use of time series analysis. J. Sensors **2017**(1), 8782131 (2017)
3. Anjali, A., Singh, V., Kumar, J.: GSSX: goldensection-based symbolic representation for reducing time series dimensionality. Int. J. Data Sci. Anal. 1–18 (2025)
4. Baltrušaitis, T., Robinson, P., Morency, L.P.: Openface: an open source facial behavior analysis toolkit. In: 2016 IEEE Winter Conference on Applications of Computer Vision (WACV), pp. 1–10. IEEE (2016)
5. Box, G.E., Cox, D.R.: An analysis of transformations. J. R. Stat. Soc. Ser. B Stat Methodol. **26**(2), 211–243 (1964)
6. Calders, T., Garboni, C., Goethals, B.: Approximation of frequentness probability of itemsets in uncertain data. In: 2010 IEEE International Conference on Data Mining, pp. 749–754 (2010)

7. Castro, N., Azevedo, P.: Multiresolution motif discovery in time series. In: Proceedings of the 2010 SIAM International Conference on Data Mining, pp. 665–676 (2010)
8. Dau, H.A., Keogh, E., Kamgar, K., Yeh, C.C.M., Zhu, Y., Gharghabi, S., Ratanamahatana, C.A., Yanping, Hu, B., Begum, N., Bagnall, A., Mueen, A., Batista, G., Hexagon-ML: The ucr time series classification archive (2018). https://www.cs.ucr.edu/~eamonn/time_series_data_2018/
9. Dau, H.A., Bagnall, A., Kamgar, K., Yeh, C.C.M., Zhu, Y., Gharghabi, S., Ratanamahatana, C.A., Keogh, E.: The UCR time series archive. IEEE/CAA J. Autom. Sin. **6**(6), 1293–1305 (2019)
10. De Paepe, D., Van Hoecke, S.: Mining recurring patterns in real-valued time series using the radius profile. In: 2020 IEEE International Conference on Data Mining (ICDM), pp. 984–989. IEEE (2020)
11. Feremans, L., Schäfer, P., Meert, W.: Motiplus and motiset: discovering the best set of motiflets in time series. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, p. TBD. Springer (2025)
12. Feremans, L., Cule, B., Goethals, B.: PETSC: pattern-based embedding for time series classification. Data Min. Knowl. Disc. **36**(3), 1015–1061 (2022)
13. Fournier-Viger, P., Wu, C.W., Gomariz, A., Tseng, V.S.: VMSP: efficient vertical mining of maximal sequential patterns. In: Advances in Artificial Intelligence: 27th Canadian Conference on Artificial Intelligence, Canadian AI 2014, Montréal, QC, Canada, May 6–9, 2014. Proceedings 27, pp. 83–94 (2014)
14. Gao, Y., Lin, J.: HIME: discovering variable-length motifs in large-scale time series. Knowl. Inf. Syst. **61**, 513–542 (2019)
15. Gomariz, A., Campos, M., Marin, R., Goethals, B.: ClaSP: An efficient algorithm for mining frequent closed sequences. In: Advances in Knowledge Discovery and Data Mining: 17th Pacific-Asia Conference, PAKDD 2013, Gold Coast, Australia, Apr 14–17, 2013, Proceedings, Part I 17, pp. 50–61 (2013)
16. Guglielmo, G., Mavromoustakos Blom, P., Klincewicz, M., Huis in't Veld, E., Spronck, P.: Blink to win: Blink patterns of video game players are connected to expertise. In: Proceedings of the 17th International Conference on the Foundations of Digital Games, pp. 1–7 (2022)
17. Kamgar, K., Gharghabi, S., Keogh, E.: Matrix profile XV: exploiting time series consensus motifs to find structure in time series sets. In: 2019 IEEE International Conference on Data Mining (ICDM), pp. 1156–1161 (2019)
18. Killeen, P., Kiringa, I., Yeap, T.: Unsupervised dynamic sensor selection for iot-based predictive maintenance of a fleet of public transport buses. ACM Trans. Internet of Things **3**(3), 1–36 (2022)
19. Latecki, L.J., Lakamper, R., Eckhardt, T.: 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, pp. 424–429 (2000)
20. Law, S.M.: STUMPY: a powerful and scalable python library for time series data mining. J. Open Source Softw. **4**(39), 1504 (2019)
21. Leite, G.d.N.P., Farias, F.C., de Sa, T.G., da Costa, A.C.A., Brennand, L.J.P., de Souza, M.G.G., Villa, A.A.O., Droguett, E.L.: A robust fleet-based anomaly detection framework applied to wind turbine vibration data. Engineering Applications of Artificial Intelligence **126**, 106859 (2023)
22. Li, Y., Lin, J., Oates, T.: Visualizing variable-length time series motifs. In: Proceedings of the 2012 SIAM International Conference on Data Mining, pp. 895–906. SIAM (2012)
23. Lin, J., Keogh, E., Wei, L., Lonardi, S.: Experiencing SAX: a novel symbolic representation of time series. Data Min. Knowl. Disc. **15**, 107–144 (2007)
24. Linardi, M., Palpanas, T.: Scalable data series subsequence matching with ULISSE. VLDB J. **29**(6), 1449–1474 (2020)
25. Linardi, M., Zhu, Y., Palpanas, T., Keogh, E.: Matrix profile goes MAD: variable-length motif and discord discovery in data series. Data Min. Knowl. Disc. **34**(4), 1022–1071 (2020)
26. Madrid, F., Imani, S., Mercer, R., Zimmerman, Z., Shakibay, N., Keogh, E.: Matrix profile XX: Finding and visualizing time series motifs of all lengths using the matrix profile. In: 2019 IEEE International conference on big knowledge (ICBK), pp. 175–182. IEEE (2019)
27. Mueen, A., Chavoshi, N.: Enumeration of time series motifs of all lengths. Knowl. Inf. Syst. **45**(1), 105–132 (2015)
28. Patel, P., Keogh, E., Lin, J., Lonardi, S.: Mining motifs in massive time series databases. In: 2002 IEEE International Conference on Data Mining, 2002. Proceedings, pp. 370–377 (2002)
29. Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., Hsu, M.C.: Mining sequential patterns by pattern-growth: the PrefixSpan approach. IEEE Trans. Knowl. Data Eng. **16**(11), 1424–1440 (2004)
30. Raza, A., Kramer, S.: Accelerating pattern-based time series classification: a linear time and space string mining approach. Knowl. Inf. Syst. **62**(3), 1113–1141 (2020)
31. Rotman, S.J., Cule, B., Feremans, L.: Efficiently mining frequent representative motifs in large collections of time series. In: 2023 IEEE International Conference on Big Data (BigData), pp. 66–75 (2023)
32. Rotman, S.J., Guglielmo, G., Cule, B., Klincewicz, M.: Expertise prediction of tetris players using eye tracking information. In: International Symposium on Intelligent Data Analysis, pp. 305–317. Springer (2025)
33. Schäfer, P., Leser, U.: Motiflets: simple and accurate detection of motifs in time series. Proc. VLDB Endow. **16**(4), 725–737 (2022)
34. Scheltinga, B.L., Reenalda, J., Buurke, J.H., Kok, J.N.: Development of models to quantify training load in outdoor running using inertial sensors. In: International Symposium on Intelligent Data Analysis, pp. 17–27. Springer (2025)
35. Schwenke, L., Bloemheuvel, S., Atzmueller, M.: Identifying informative nodes in attributed spatial sensor networks using attention for symbolic abstraction in a GNN-based modeling approach. In: The International FLAIRS Conference Proceedings, vol. 36 (2023)
36. Senin, P., Lin, J., Wang, X., Oates, T., Gandhi, S., Boedihardjo, A.P., Chen, C., Frankenstein, S.: Grammarviz 3.0: interactive discovery of variable-length time series patterns. ACM Trans. Knowl. Discovery Data (TKDD) **12**(1), 1–28 (2018)
37. Srikant, R., Agrawal, R.: Mining sequential patterns: generalizations and performance improvements. In: International Conference on Extending Database Technology, pp. 1–17 (1996)
38. Torkamani, S., Lohweg, V.: Survey on time series motif discovery. Wiley Interdiscip. Rev. Data Min. Knowl. Discov. **7**(2), e1199 (2017)
39. Truong, C.D., Anh, D.T.: A fast method for motif discovery in large time series database under dynamic time warping. In: Knowledge and Systems Engineering: Proceedings of the Sixth International Conference KSE 2014, pp. 155–167. Springer (2015)
40. Truong, C.D., Anh, D.T.: A novel clustering-based method for time series motif discovery under time warping measure. Int. J. Data Sci. Anal. **4**(2), 113–126 (2017)
41. van den Biggelaar, L., Schouten, R.M., de Bie, A., Bouwman, R.A., Duivesteijn, W.: Characterizing the risk of atrial fibrillation in cardiac patients with exceptional electrocardiogram phenotypes. In: Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2, pp. 4925–4934 (2025)
42. van Onsem, M., Ledoux, V., Mélange, W., Dreesen, D., Van Hoecke, S.: Variable length motif discovery in time series data. IEEE Access (2023)
43. van Wesenbeeck, D., Yurtman, A., Meert, W., Blockeel, H.: LoCoMotif: discovering time-warped motifs in time series. In: Data Mining and Knowledge Discovery, pp. 1–30 (2024)

44. Yeh, C.C.M., Zhu, Y., Ulanova, L., Begum, N., Ding, Y., Dau, H.A., Silva, D.F., Mueen, A., Keogh, E.: 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), pp. 1317–1322 (2016)
45. Zhang, M., Wang, P., Wang, W.: Efficient consensus motif discovery of all lengths in multiple time series. In: International Conference on Database Systems for Advanced Applications, pp. 540–555 (2022)
46. Zhong, S., Mueen, A.: MASS: distance profile of a query over a time series. In: Data Mining and Knowledge Discovery pp. 1–27 (2024)
47. Zhu, Y., Yeh, C.C.M., Zimmerman, Z., Kamgar, K., Keogh, E.: Matrix profile XI: SCRIMP++: time series motif discovery at interactive speeds. In: 2018 IEEE International Conference on Data Mining (ICDM), pp. 837–846 (2018)
48. Zhu, Y., Zimmerman, Z., Senobari, N.S., Yeh, C.C.M., Funning, G., Mueen, A., Brisk, P., Keogh, E.: 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), pp. 739–748 (2016)