2021 IEEE International Conference on Data Mining (ICDM) | 978-1-6654-2398-4/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICDM51629.2021.00054

# Multi-way Time Series Join on Multi-length Patterns

Md Parvez Mollah*, Vinicius M. A. Souza*†, and Abdullah Mueen*
*Department of Computer Science, University of New Mexico, Albuquerque, USA
†Graduate Program in Informatics, Pontifícia Universidade Católica do Paraná, Curitiba, Brazil
Email: *{parvez, vinicius, mueen}@unm.edu, †vinicius@ppgia.pucpr.br

***Abstract*—This paper introduces a new pattern mining task that considers aligning or joining a set of time series based on an arbitrary number of subsequences (i.e., patterns) with arbitrary lengths. Joining multiple time series along common patterns can be pivotal in clustering and summarizing large time series datasets. An exact algorithm to join hundreds of time series based on multi-length patterns is impractical due to the high computational costs. This paper proposes a fast algorithm named MultiPAL to join multiple time series at interactive speed to summarize large time series datasets. The algorithm exploits Matrix Profiles of the individual time series to enable a greedy search over possible joins. The algorithm is orders of magnitude faster than the exact solution and can utilize hundreds of Matrix Profiles. We evaluate our algorithm for sequential mining on data from various real-world domains, including power management and bioacoustics monitoring.**

***Keywords*-Time series, multi-length patterns, join, motifs**

## I. Introduction

Time series join is a data mining task that discovers the highly similar subsequence (i.e., a pattern) between two large time series. Joins provide useful information about the data synchrony and can help data analysts better understand the correlations of sequential phenomena [1]. Surprisingly, the solutions for this problem are limited to a pair of time series and a single aligning/joining pattern [1]–[4]. In this work, we introduce the problem of joining multiple time series based on an arbitrary number of subsequences with variable lengths. Joining multiple time series along highly similar subsequences can be pivotal in clustering and summarizing large time series datasets on various domains.

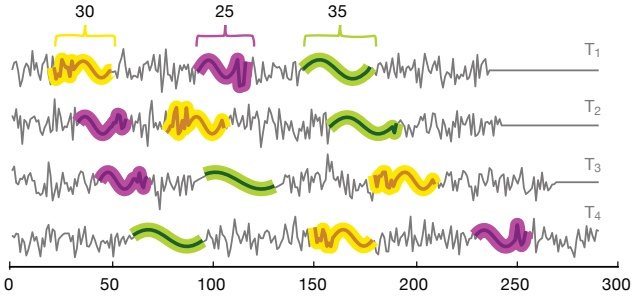



Fig. 1. Four time series with planted sinusoidal patterns. Each sinusoid is of different length. A multi-way join algorithm groups the **smallest** sinusoids, the **medium** sinusoids and the **largest** sinusoids together.

To illustrate multi-way join, consider the four time series shown in Fig. 1. In this example, each time series has three sinusoidal patterns artificially planted in random noise, each pattern with a different length. The multi-way join algorithm seeks to discover the highly similar subsequences among the series without prior information about the *number* and the *length* of the patterns. In the example, the algorithm correctly groups the **smallest** sinusoidal patterns, the **medium** patterns, and the **largest** patterns. It is interesting to note that we can consider each discovered pattern to join the four time series uniquely. Hence, in effect, there are three different multi-way (i.e., four-way) joins, as illustrated in Fig. 2.

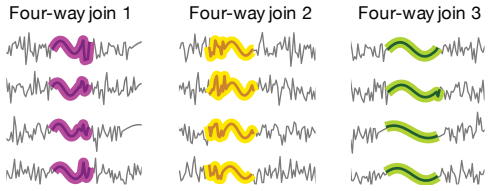



Fig. 2. Three individual joins obtained from the multi-length patterns discovered in Fig. 1.

An exact algorithm to join hundreds of time series based on multiple patterns is impractical due to the high computational costs. As explained in this paper, the time complexity of multi-way join on time series is exponential to the number of time series. On a dataset with one hundred time series with 100,000 observations in each, a desktop machine would spend about many years to compute multi-way joins of various number of time series and lengths. Moreover, the space complexity of such an algorithm is beyond manageable by any single computing system. Hence, data mining practitioners would benefit from a realistic algorithm for joining multiple time series in a flexible and robust manner.

We develop a fast algorithm to join multiple time series at a reasonable speed to summarize a large set of time series. Our algorithm **MultiPAL** (*Multiple Patterns ALignments*) exploits Matrix Profiles [2] of all pairs of time series to enable a greedy search over possible multi-way joins. The algorithm is orders of magnitude faster than the exact solution and can process hundreds (if not thousands) of Matrix Profiles. The algorithm generates meaningful patterns, which achieve better clustering performance compared to the patterns generated by existing algorithms. We apply the algorithm to meaningfully

978-1-6654-2398-4/21/$31.00 ©2021 IEEE
DOI 10.1109/ICDM51629.2021.00054

summarize real-world datasets to domain experts in the fields of power engineering and bioacoustics. The remaining sections of this article provide background definitions, related works, algorithmic descriptions and experimental results.

## II. Definitions and notation

In this section, we define the necessary terms and concepts to describe the algorithm in Section IV.

**Definition 1 (time series):** A time series $T$ of length $m$ is an ordered sequence of real numbers $t_i$ measured in equally spaced time, in which $T = (t_1, t_2, \ldots, t_m)$.

**Definition 2 (multiple time series):** A dataset $\mathcal{D} = \{T_1, \ldots, T_n\}$ with multiple time series is a collection of $n$ independent series $T$ ($n \geq 2$). Here, each time series can have a different number of observations $m$, and can be asynchronous to other time series.

**Definition 3 (subsequence):** A subsequence $S_{i,L}$ is a continuous segment of length $L$ from a time series $T$ starting from position $i$. $S_{i,L} = (t_i, t_{i+1}, \ldots, t_{i+L-1})$, where $1 \leq i \leq m-L+1$. For a time series of length $m$, there can be a total of $\frac{m(m+1)}{2}$ subsequences of all possible lengths. For $n$ time series with the same length $m$, the total number of subsequences can be $n \times \frac{m(m+1)}{2}$. In this paper, the word pattern refers to a group of highly similar subsequences.

**Definition 4 (similarity join):** Given two time series $T_a$ and $T_b$ and a subsequence length $L$, the similarity join (i.e., two-way join) identifies the *most similar* pair of subsequences, one from $T_a$ and the other from $T_b$. When time series are recorded at high-precision (i.e., 32-bit), usually, there is exactly one closest pair of subsequences. The similarity between the subsequences can be calculated by the Euclidean distance, Dynamic Time Warping, Pearson's Correlation Coefficient, among many others distance measures. In this paper, we use Pearson's Correlation Coefficient, as defined below.

$$corr(x, y) = 1 - \frac{\sum_{i=1}^{L}(X_i - Y_i)^2}{2L}$$

where $X_i = \frac{x_i - \mu_x}{\sigma_x}$ for $i = 1, 2, \ldots, L$, $\mu_x$ is the mean and $\sigma_x$ is the sample standard deviation of a subsequence $x$.

**Definition 5 (multi-way join):** A multi-way join $(N, L, C)$ of a set of time series ($n \geq 2$) is a set of $N$ subsequences of length $L$, one from each time series, and the subsequences are similar to each other with a ***minimum*** correlation coefficient, $C$. For example, in Figure 1, the **smallest** sinusoids represent a $(4, 25, 0.75)$ four-way join, and the **largest** sinusoids show a $(4, 35, 0.63)$ four-way join.

**Definition 6 (non-trivial multi-way join):** A multi-way join $(N_2, L_2, C_2)$ is non-trivial if none of the $N_2$ joining subsequences overlaps with any of the $N_1$ joining subsequence in another multi-way join $(N_1, L_1, C_1)$ on the same time series.

Given the above definitions, we are now in position to define the problem of finding multi-way joins on multiple time series.

**General Problem Definition:** *Given a collection of $n$ time series, find $k$ non-trivial multi-way joins $(N \leq n, L, C)$, maximizing $N$, $L$ and $C$.*

Ideally, with infinite resources and time, a data mining practitioner would want to maximize all the variables $k, N, L$ and $C$. There is a natural limit on human capacity in processing outputs generated by mining algorithms, hence, we define $k$ as a user given input. However, maximizing all of $N$, $L$ and $C$ entails searching a mammoth space of multi-way joins. More specifically, for each value of $N$, there are $\binom{n}{N}$ combinations of time series. For each of these combinations, the general problem requires the largest length $L$ and the largest minimum correlation $C$.

In practice, we can reasonably set a priority order among these variables as $C$ being the most important because dissimilar (i.e., low $C$) subsequences do not form any pattern. $N$ is the next important variable because (N+1)-way join is non-trivial to find given an $N$-way join. We consider $L$ as the least sensitive variable and propose to search over a small set of pattern lengths depending on the number of joins a user wants. This leads to a modified problem definition for multi-way join on multiple time series.

**Practical Problem Definition:** *Given a collection of $n$ time series, find at most $k$ non-trivial multi-way joins $(N \leq n, L, C)$ for $k$ different $L$, maximizing $N$ and $C$.*

According to our problem definition, there are $k = 3$ multi-way joins in the example of Figure 1. Those are $(4, 25, 0.75)$, $(4, 30, 0.69)$ and $(4, 35, 0.63)$, all with variable lengths. In this paper, we solve the practical problem both *exactly* and *approximately*.

## III. Related work

Unsupervised temporal pattern mining is a very well-studied area of research. Specific patterns include motifs [5], discords [2], uShapelets [6], and time series join [1]. However, most of these patterns are introduced assuming the pattern's length is fixed and given as an input to the algorithm. Relaxing this assumption, methods have been proposed to discover variable-length motifs [7]–[9]. These methods work on one long time series and exploit similarity among overlapping subsequences to reduce computational time. In contrast, multidimensional motif discovery [10]–[12] focuses on multiple time series with synchronous patterns across dimensions. However, true relaxation is only achieved when the user can *discover patterns of any length from any number of time series*. To the best of our knowledge, this work is the first attempt to solve this grand pattern mining problem.

Yeh et al. [11] introduced an algorithm (mSTAMP) to find motifs over multidimensional time series. The main difference is that the patterns discovered by mSTAMP must be synchronized in time across all the dimensions. mSTAMP only considers patterns with a fixed length. MultiPAL produces patterns that are asynchronous, multi-length, and independent of original dimensionality.

A more related task to ours is two-way *time series join* [1]. The main goal is to join two long time series based on the most correlated segment from each one. The time series can be joined at any location and for arbitrary length. The task's

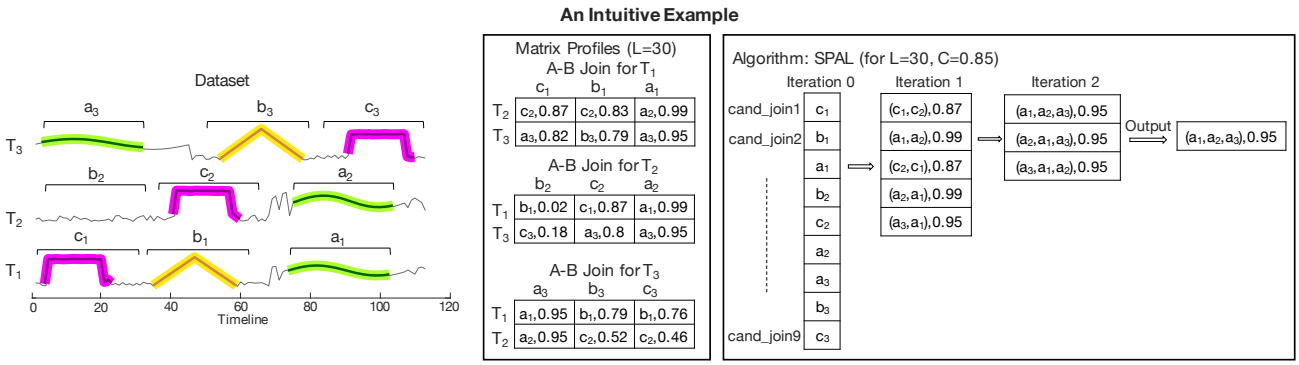



Fig. 3. An illustration of the proposed MultiPAL algorithm generating the best multi-way join for a length $L = 30$.

challenge is the expensive computational cost to compare a massive number of subsequences. Given series with length $m$, a naive algorithm that computes all the possible pairs of segments of all the lengths requires $O(m^4)$ to find the most similar subsequences between the pair of series [1].

There is a considerable effort to reduce the time complexity for searching correlated subsequences in a pair of time series [1]–[4]. The extension of this problem for multiple series brings new challenges and is still an unexplored task in the literature. Most existing algorithms that look at aligning and/or joining multiple time series are based on pattern similarity above or below a learned threshold, such as U-Shapelets [1]. In contrast, we join or align multiple time series based on a sequence of nearest neighbors, exploiting the Matrix Profiles.

## IV. Multi-way Multi-length Join

### A. Intuitive example

We propose an Apriori-like [13] approach to grow join candidates while maximizing support and minimum similarity. We give a toy example in Figure 3 to illustrate how the proposed MultiPAL algorithm (Algorithm 4) finds the best multi-way join, maximizing $N$ and $C$, from a set of three time series ($T_1$, $T_2$, $T_3$) for a length $L = 30$. For simplicity, we consider only 3 subsequences from each time series. In Figure 3, they are represented as $a_i$, $b_i$, and $c_i$ in each $T_i$. Ideally, the output should produce $(a_1, a_2, a_3)$ for length 30, $(b_1, b_3)$ for length 25, and $(c_1, c_2, c_3)$ for length 15, given its similarities.

At first, we compute the A-B join between each pair of time series $T_i$ and $T_j$ $(i \neq j)$ for $L = 30$, i.e., the most similar subsequence in $T_j$ for each subsequence in $T_i$, using SCAMP algorithm [14]. From the table A-B join for $T_1$ in Figure 3, we see that $a_2 \in T_2$ and $a_3 \in T_3$ are the most similar to $a_1 \in T_1$ with correlations 0.99 and 0.95, respectively. For simplicity, let us assume a minimum correlation coefficient of $C = 0.85$ for the A-B joins. In Section IV-E we discuss in detail how the algorithm automatically set the threshold $C$.

We start with the set of all subsequences of length $L$ as candidates joins. In this example, we consider only nine candidates, $\{cand\_join1, cand\_join2, ..., cand\_join9\}$, as shown at $Iteration$ 0 in the Figure 3.

At the first iteration, we take each $cand\_join$ and expand it by adding one subsequence in each iteration. We find the most similar subsequence from $U$ to the last added subsequence in $cand\_join$. Here, $U$ is the set of time series that are not included in $cand\_join$. For example, consider $cand\_join1$ in Figure 3. The last added subsequence in $cand\_join1$ is $c_1$ and $U = \{T_2, T_3\}$. We look into the $c_1$ column of the table A-B join for $T_1$ and find that the subsequence $c_2 \in T_2$ is the most similar to $c_1$ with correlation 0.87 among all time series in $U$. Since $0.87 \geq C$, we add $c_2$ in $cand\_join1$ at $Iteration$ 1. In the process, we drop the candidates $cand\_join2$, $cand\_join4$, $cand\_join8$, and $cand\_join9$ at $Iteration$ 1 because their nearest neighbors have a correlation below the threshold $C$. Hence, we stop expanding these joins. At the end of $Iteration$ 1, we get five candidate multi-way joins, each with two subsequences in it.

The process of growing the candidate joins continues until no more subsequences can be added in any of the remaining joins. In the given example, the algorithm stops at $Iteration$ 2 as each of the remaining candidate multi-way joins have subsequences from all three time series. The candidate multi-way joins at the last iteration have the maximum number of subsequences in them. Thus, $N$ is maximized. The algorithm outputs the multi-way join which has the largest minimum correlation between consecutive pairwise subsequences. For example, for the candidate join $(a_1, a_2, a_3)$, the pairwise consecutive correlations include correlations between $(a_1, a_2)$ and $(a_2, a_3)$. Each of the three candidate multi-way joins at $Iteration$ 2 in Figure 3 has 0.95 as the minimum pairwise consecutive correlation. We choose the first of them as the output.

### B. Exact algorithm

First, we present an exact algorithm (Algorithm 1) that finds non-trivial joins maximizing $N$ and $C$ for a given set of $k$ different lengths, $\mathcal{L}$. The algorithm works in two phases, as follows:

In phase 1 (lines 1-13), the algorithm generates one non-trivial join, maximizing the number of time series and the minimum correlation coefficient, for each subsequence length

$L \in \mathcal{L}$. To do so, the algorithm takes each subsequence length $L$ and sets the minimum correlation coefficient $C$ as the $95^{th}$ percentile of the distribution of the highest similarities (i.e., nearest neighbor distances) over all possible subsequences (lines 2-4). The reason for this choice of $C$ is described in Section IV-E. To compute the nearest neighbors of all possible subsequences of length $L$, we exploit the Matrix Profile algorithm [14].

Algorithm 2 is called for each subsequence $S_{a,i,L}$ in each time series $T_a \in \mathcal{D}$ (lines 6-10). Algorithm 2 finds all non-trivial multi-way joins containing the given subsequence $S_{a,i,L}$ by recursively calling itself. The algorithm stores the generated joins for the subsequence length $L$ in $cand\_joins$ (line 8). At line 12, the algorithm picks the *best* join from $cand\_joins$, consisting of the maximum number of time series and the largest minimum correlation coefficient, and appends the best join to the set $single\_joins$. In phase 2 (lines 14-15), all multi-way joins in $single\_joins$ are merged to get at most $k$ multi-way multi-length joins by calling Algorithm 3. The obtained result, i.e., $multi\_joins$ is returned as the output at line 15.

For each unexplored time series $T_b$, Algorithm 2 finds the subsequence in $T_b$ that is the most similar to the subsequences in the $Candidate$ join (i.e., reverse nearest neighbors in lines 5-17). To do so, at first, the correlation coefficient between each subsequence in $Candidate$ and all subsequences in $T_b$ is computed using the MASS algorithm [15] (line 8). In the loop at lines 9-15, for each subsequence $S_{b,j,L}$ in $T_b$, if the correlation coefficients between $S_{b,j,L}$ and all subsequences in $Candidate$ maintain the correlation coefficient threshold $C$, the minimum among them is stored in $min\_corr$ array. The subsequence with the largest minimum correlation (i.e., $max(min\_corr)$) is added in $Candidate$, and the algorithm calls itself with this new candidate (lines 18-21). Finally, if no subsequence is found to be added after iterating all unexplored time series, the algorithm appends $Candidate$ to the result set $cand\_joins$ (lines 23-26).

Algorithm 3 generates multi-patterns joins by combining the multi-way joins obtained in phase 1 of the Exact algorithm, i.e., Algorithm 1. Inside the loop at line 1, the algorithm finds all $l$-patterns multi-way joins by choosing all possible $l$-size subsets from $\mathcal{P}(single\_joins)$, i.e., the power set of multi-way joins set, and stores them in $Candidates$ (lines 3-7). If some $l$-patterns joins are found, they are sorted by the number of joined time series (higher number of series first) (line 9). If multiple joins have the same number of time series, they are sorted by the average pairwise overlap score (lower score first), as described in Section IV-F. Next, the best $l$-patterns join is saved in $multi\_joins$ (line 10) and the algorithm returns $multi\_joins$ (line 14).

Next, we analyze the time complexity of the Exact algorithm. Assume we have $n$ time series with length $m$. At first, we describe the runtime of Algorithm 2 and 3. The MASS algorithm at line 8 of Algorithm 2 takes $O(m \log m)$ time [15]. Next, the loop in line 4 runs $(n-t)$ times where $t$ is the number of subsequences in the $Candidate$ join. Thus the time complexity of lines 4-16 is $O((n-t)tm \log m)$. At line 20, Algorithm 2 calls itself recursively, which runs at most $(n-t)$ times because the number of subsequences in $Candidate$ grows by one at each recursive call of Algorithm 2. Hence, the time complexity of Algorithm 2, $T(n) = (n-1)(T(n-1) + m \log m)$, which is $O(n! m \log m)$.

The loops in line 1 and 3 of Algorithm 3 together take

**Algorithm 1** ExactMultipleJoin($\mathcal{D}$,$m$,$k$,$\mathcal{L}$)

**Require:** A dataset $\mathcal{D}$ with $n$ time series, time series length $m$, a value $k$, a set of subsequence lengths $\mathcal{L}$
**Ensure:** A set of multi-way joins
1: $single\_joins \leftarrow \emptyset$
2: **for** each subsequence length $L \in \mathcal{L}$ **do**
3: $mp \leftarrow$ load matrix profiles for all pairs of time series for length $L$
4: $C \leftarrow 95^{th}$ percentile of $mp$
5: $cand\_joins \leftarrow \emptyset$
6: **for** each time series $T_a$ in $\mathcal{D}$ **do**
7: **for** each subsequence $S_{a,i,L}$ in $T_a$ **do**
8: $cand\_joins \leftarrow FindJoins(\{S_{a,i,L}\}, \mathcal{D}, m, L, C, cand\_joins)$
9: **end for**
10: **end for**
11: $cand\_joins \leftarrow sort(cand\_joins)$
12: $single\_joins \leftarrow single\_joins \cup cand\_joins\{1\}$
13: **end for**
14: $multi\_joins \leftarrow MergeSingleJoins(\mathcal{D}, m, k, single\_joins)$
15: **return** $multi\_joins$

**Algorithm 2** FindJoins($Candidate$,$\mathcal{D}$,$m$,$L$,$C$,$cand\_joins$)

1: $candidate\_size \leftarrow size(Candidate)$
2: $total\_subseqs \leftarrow m - L + 1$
3: $found \leftarrow false$
4: **for** each time series $T_b$ not participating in $Candidate$ **do**
5: $min\_corr(1 : total\_subseqs) \leftarrow 1.0$
6: **for** $j \leftarrow 1$ to $candidate\_size$ **do**
7: $S_{a,i,L} \leftarrow Candidate(j)$
8: $corr(1 : total\_subseqs) \leftarrow MASS(T_b, S_{a,i,L})$
9: **for** $k \leftarrow 1$ to $total\_subseqs$ **do**
10: **if** $corr(k) \geq C_{min}$ **then**
11: $min\_corr(k) \leftarrow min(min\_corr(k), corr(k))$
12: **else**
13: $min\_corr(k) \leftarrow -\infty$
14: **end if**
15: **end for**
16: **end for**
17: $(mx, id) \leftarrow max(min\_corr)$
18: **if** $mx > 0$ **then**
19: $found \leftarrow true$
20: $cand\_joins \leftarrow FindJoins(Candidate \cup \{S_{b,id,L}\}, \mathcal{D}, m, L, C, cand\_joins)$
21: **end if**
22: **end for**
23: **if** $found = false$ **then**
24: $cand\_joins \leftarrow cand\_joins \cup \{Candidate\}$
25: **end if**
26: **return** $cand\_joins$

**Algorithm 3** MergeJoins($\mathcal{D}$,$m$,$k$,$single\_joins$)

1: **for** $l \leftarrow k$ to 1 **do**
2: $Candidates \leftarrow \emptyset$
3: **for** each subset $joins\_set$ of size $l$ in $\mathcal{P}(single\_joins)$ **do**
4: $T\_ids \leftarrow$ set of time series indices $\in$ all joins in $joins\_set$
5: $S \leftarrow$ set of subsequences of $T\_ids$ in $joins\_set$
6: $Candidates \leftarrow Candidates \cup \{(T\_ids, S)\}$
7: **end for**
8: **if** $Candidates \neq \emptyset$ **then**
9: $sort(Candidates)$
10: $multi\_joins \leftarrow Candidates(1)$
11: $break$
12: **end if**
13: **end for**
14: **return** $multi\_joins$

$\binom{l_s}{1} + \binom{l_s}{2} + ... + \binom{l_s}{k} = O(2^{l_s})$ where $l_s$ is the size of the set $single\_joins$, i.e., $l_s = |\mathcal{L}|$. Lines 4 to 6 can be done in $O(n)$ time, which yields a total time complexity of $O(n2^{|\mathcal{L}|})$ for Algorithm 3. As the cost of computing matrix profile for a pair of time series is quadratic $O(m^2)$ [2], the total cost of precomputing matrix profiles for all pairs is $O((nm)^2)$. Lines 2-13 of Algorithm 1 require $O(|\mathcal{L}|n!nm^2 \log m)$ time because Algorithm 2 is called for each subsequence. We can see $O(n2^{|\mathcal{L}|}) << O((nm)^2) << O(|\mathcal{L}|n!nm^2 \log m)$. Therefore, the total cost of the Exact algorithm is $O(|\mathcal{L}|n!nm^2 \log m)$, clearly an exponential algorithm.

### C. Greedy algorithm

The time complexity of the Exact algorithm grows exponentially as the number of time series $n$ increases, making it impractical in real life. To reduce the exponential search space, we propose a greedy heuristic solution to find multiple different non-trivial joins.

- First, while adding a new subsequence in a candidate multi-way join, we only consider the correlation between the new subsequence and the last added subsequence in the candidate. In other words, if a multi-way join has $n$ subsequences, we guarantee $(n-1)$ pairs of subsequences are similar to each other with a minimum correlation coefficient $C$, instead of ensuring all $\frac{n(n-1)}{2}$ pairs. This choice might decrease the joins' quality; however, it enables us to apply the next greedy optimization.
- Second, we do not need to use the MASS algorithm and iterate over all the subsequences in a time series to find the most similar subsequence to be added in the candidate join. Instead, we take the subsequence which is the most similar to the last added subsequence in the candidate join by exploiting the Matrix Profiles of all pairs of time series.
- Third, we use the breadth-first search strategy (BFS) instead of the depth-first search strategy (DFS) to generate the candidate joins. In each level of BFS, the algorithm grows each candidate join by adding a subsequence in it. BFS prevents our algorithm from running out of stack memory for large $n$ but presents the challenge of running short of main memory. To deal it, we consider a fixed number of candidate joins in each level of BFS, and we call this parameter as memory threshold $M_t$.

Algorithm 4 describes our solution, which takes a set of $n$ time series each with length $m$, a value $k$ representing the number of different multi-way joins that need to be produced, a set of subsequence lengths $\mathcal{L}$, and a memory threshold $M_t$ as inputs and outputs up to $k$ different multi-way joins. MultiPAL works in two phases, as follows.

In phase 1 (lines 1-12), the algorithm generates multi-way joins for each subsequence length $L \in \mathcal{L}$ and stores them in the set $single\_joins$. At line 3, the precomputed all pairs of matrix profiles and matrix profiles indices of the current subsequence length are loaded in $mp$ and $mp_I$, respectively. Similar to the Exact algorithm, we set the minimum correlation coefficient $C$ to the $95^{th}$ percentile of the distribution of the highest similarities (i.e., nearest neighbor distances) over all possible subsequences (line 4). Inside the loop at line 5, Algorithm 5 is invoked to find the *best* multi-way join, maximizing the number of time series and the minimum consecutive pairwise correlation, for the current $L$. If a join is found, it is appended to $single\_joins$ and Algorithm 5 is invoked again to get another multi-way join from the time series that are not joined yet (lines 6-10). Otherwise, the algorithm moves on to work with the next subsequence length.

In phase 2 (lines 13-14), MultiPAL follows the same process as the Exact algorithm, i.e., using Algorithm 3 to generate multi-patterns joins by combining the multi-way joins obtained in phase 1. The obtained result, i.e., $multi\_joins$ is returned as the output at line 14.

Algorithm 5 is used to find the best multi-way join for a subsequence length $L$. At first, the algorithm generates candidate multi-way joins containing two subsequences using the precomputed matrix profiles $mp$ and $mp_I$ (lines 1-17). To do so, the algorithm takes each time series $T_a$ (line 3) from the set of time series $U$ that are not joined yet and each subsequence $S_{a,i,L}$ from $T_a$ (line 4). At lines 6-7, for each time series $T_b$ in $\{U - T_a\}$, the algorithm gets the subsequence $S_{b,j,L}$, which is the nearest neighbor of $S_{a,i,L}$ in $T_b$, by looking up the entry $mp_I(a,b,i)$. The correlation between $S_{a,i,L}$ and $S_{b,j,L}$, i.e., $mp(a,b,i)$ is saved in $corr$ at line 8. If any of these two subsequences contain trivial matches, in other words, if more than $10\%$ of the subsequence is taken in another join, this pair of subsequences is not considered as a candidate. Otherwise, if $corr$ is greater or equal to $mx\_corr$ (initially set to the minimum correlation coefficient $C$ at line 5), the subsequence $S_{b,j,L}$ is saved in $best\_subseq$ along with the correlation $corr$ in $mx\_corr$ (line 10). After iterating all time series in $\{U - T_a\}$, the subsequence stored in $best\_subseq$ is the most similar to $S_{a,i,L}$. Hence, these two subsequences form a candidate multi-way join which is appended to the candidate joins set $Candidates$ (line 13-15). At line 19, the algorithm sorts $Candidates$ by the correlation between the subsequences (higher correlation first). The top $M_t$ candidate joins are chosen from $Candidates$ to be used in the next level (line 20).

The algorithm takes each join from the initial set of candidate multi-way joins and expands it by adding more subsequences. In order to do so, the algorithm starts a loop (line 23) which runs at most $(n-2)$ times as a multi-way join can contain at most $n$ subsequences, one from each time series. Inside the loop, the algorithm takes each join $candidate$ from $Candidates$ (line 25) and the last added subsequence $S_{a,i,L}$ from $candidate$ (line 26). At lines 27-34, the algorithm finds the most similar subsequence of $S_{a,i,L}$ among all the time series $T_b$ not participating in $candidate$, which is stored in $best\_subseq$. A non-empty $best\_subseq$ is added in the $candidate$ join and $candidate$ is appended to the set of new multi-way joins, i.e., $newCandidates$ (lines 35-37). After iterating all the candidate joins, a non-empty $newCandidates$ set indicates some multi-way joins are expanded. Hence, the set $Candidates$ is replaced with $newCandidates$ and the process of growing the joins is continued (lines 39-41). When

this process completes, the set $Candidates$ contains the multi-way joins with the maximum number of time series. The multi-way join with the largest minimum correlation between consecutive subsequences is returned (lines 46-47).

**Algorithm 4** MultiPAL($\mathcal{D}$,$m$,$k$,$\mathcal{L}$,$M_t$)

**Require:** A dataset $\mathcal{D}$ with $n$ time series, time series length $m$, a value $k$, a set of subsequence lengths $\mathcal{L}$, memory threshold $M_t$
**Ensure:** A set of multi-pattern joins
1: $single\_joins \leftarrow \emptyset$
2: **for** each subsequence length $L \in \mathcal{L}$ **do**
3: $mp, mp_I \leftarrow$ load matrix profiles for all pairs of time series for length $L$
4: $C \leftarrow 95^{th}$ percentile of $mp$
5: **while** $true$ **do**
6: $cand\_join \leftarrow SPAL(n, m, L, mp, mp_I, C, M_t, A)$
7: **if** $cand\_join = \emptyset$ **then**
8: $break$
9: **end if**
10: $single\_joins \leftarrow single\_joins \cup cand\_join$
11: **end while**
12: **end for**
13: $multi\_joins \leftarrow MergeSingleJoins(\mathcal{D}, m, k, single\_joins)$
14: **return** $multi\_joins$

**Algorithm 5** SPAL($n$,$m$,$L$,$mp$,$mp_I$,$C$,$M_t$)

1: $U \leftarrow$ set of time series not joined yet
2: $Candidates \leftarrow \emptyset$
3: **for** each $T_a$ in $U$ **do**
4: **for** each subsequence $S_{a,i,L}$ in $T_a$ **do**
5: $best\_subseq, mx\_corr \leftarrow \emptyset, C$
6: **for** each $T_b$ in $\{U - T_a\}$ **do**
7: $S_{b,j,L} \leftarrow mp_I(a, b, i)$ // nearest neighbor of $S_{a,i,L}$ in $T_b$
8: $corr \leftarrow mp(a, b, i)$ // correlation between $S_{a,i,L}$ and $S_{b,j,L}$
9: **if** $S_{a,i,L}$ and $S_{b,j,L}$ do not contain trivial matches and $corr >= mx\_corr$ **then**
10: $best\_subseq, mx\_corr \leftarrow S_{b,j,L}, corr$
11: **end if**
12: **end for**
13: **if** $best\_subseq \neq \emptyset$ **then**
14: $Candidates \leftarrow Candidates \cup \{\{S_{a,i,L}, best\_subseq\}\}$
15: **end if**
16: **end for**
17: **end for**
18: **if** $size(Candidates) > M_t$ **then**
19: $Candidates \leftarrow sort(Candidates)$
20: $Candidates \leftarrow Candidates(1 : M_t)$
21: **end if**
22: $iteration \leftarrow 3$
23: **while** $iteration \leq n$ **do**
24: $newCandidates \leftarrow \emptyset$
25: **for** each $candidate$ in $Candidates$ **do**
26: $S_{a,i,L} \leftarrow$ last added subsequence in $candidate$
27: $best\_subseq, mx\_corr \leftarrow \emptyset, C$
28: **for** each time series $T_b$ not participating in $candidate$ **do**
29: $S_{b,j,L} \leftarrow mp_I(a, b, i)$ // nearest neighbor of $S_{a,i,L}$ in $T_b$
30: $corr \leftarrow mp(a, b, i)$ // correlation between $S_{a,i,L}$ and $S_{b,j,L}$
31: **if** $S_{b,j,L}$ do not contain trivial matches and $corr >= mx\_corr$ **then**
32: $best\_subseq, mx\_corr \leftarrow S_{b,j,L}, corr$
33: **end if**
34: **end for**
35: **if** $best\_subseq \neq \emptyset$ **then**
36: $newCandidates \leftarrow newCandidates \cup \{candidate \cup \{best\_subseq\}\}$
37: **end if**
38: **end for**
39: **if** $newCandidates \neq \emptyset$ **then**
40: $iteration \leftarrow iteration + 1$
41: $Candidates \leftarrow newCandidates$
42: **else**
43: $break$
44: **end if**
45: **end while**
46: $Candidates \leftarrow sort(Candidates)$
47: **return** $Candidates(1)$

### D. Complexity analysis

In Algorithm 5, lines 3 and 4 consider each subsequence of each time series in $U$, and line 6 takes each time series from $\{U - T_a\}$. Since the total number of subsequences in a time series is $m - L + 1$ where $L$ is the subsequence length, lines 3-6 need $O(n^2 m)$ time. The check for trivial matches can be done in constant time by precomputing and caching results in memory. So, the lines 3-17 take $O(n^2 m)$ time to finish. Sorting the $Candidates$ at line 19 takes $O(nm \log(nm))$ since each subsequence can form at most one multi-way join. The loop at line 23 runs $O(n)$ times. Since at most $M_t$ number of candidate joins can be generated in each level, lines 25-38 require $O(nM_t \log(M_t))$ time. The cost of computing lines 23-46 is $O(n^2 M_t \log(M_t))$. Therefore, the overall worst case time complexity of Algorithm 5 is $O(n^2 m + nm \log(nm) + n^2 M_t \log(M_t))$, i.e., $O(n^2 M_t \log(M_t))$.

The cost of precomputing matrix profiles for all pairs of time series is $O((nm)^2)$. For each $L \in \mathcal{L}$, Algorithm 4 (MultiPAL) iteratively calls Algorithm 5 to find the best multi-way joins from the unused time series. Since a multi-way join for a length $L$ can have one subsequence from each time series and the number of time series is maximized in a join, Algorithm 5 is called a constant number of times in the loop at line 6. Hence, lines 1-12 run in $O((nm)^2 + n^2 M_t \log(M_t))$ considering the size of $\mathcal{L}$ is small. Algorithm 3 is called at line 13. As we described in Section IV-B, the cost of Algorithm 3 is $O(n2^{l_s})$, where $l_s$ is the size of $single\_joins$, i.e., the number of multi-way joins found in total. The value of $l_s$ depends on the number of times Algorithm 5 is called, which is constant. In other words, $l_s$ is roughly $2|\mathcal{L}| \sim 3|\mathcal{L}|$. We can see $O(n2^{l_s}) << O((nm)^2 + n^2 M_t \log(M_t))$. Therefore, the total cost of MultiPAL is $O((nm)^2 + n^2 M_t \log(M_t))$.

### E. Relationship between length and correlation coefficient

Most existing works consider a linear relationship between length and correlation coefficient. The concept of length-normalized correlation is often used in the literature to compare matches of different lengths [6], [16], [17]. However, we argue that this assumption is detrimental for multi-length joins.

To demonstrate our argument, we calculate the distribution of the correlation coefficients obtained by Matrix Profiles of various lengths $L$ considering the Power dataset (see Sect. VI-A). The correlations distribution given $L = (672, 192, 96, 20)$ are shown in Figure 4(left). Figure 4(right) shows the $min$, $median$, $mean$, $max$ and $95^{th} percentile$ of each distribution. Note that any of these correlations can be in our algorithm's final multiple joins. We observe that the relationship is not linear. Moreover, the commonly used maximum correlation (i.e., motifs) shows discontinuity. It is worth mentioning that this observation also holds on other data.

Based on this observation, we propose to empirically choose the minimum correlation coefficient for a given length. Identify the $95^{th} percentile$ point from the Matrix Profiles distribution for each length and use it as the minimum correlation coefficient.

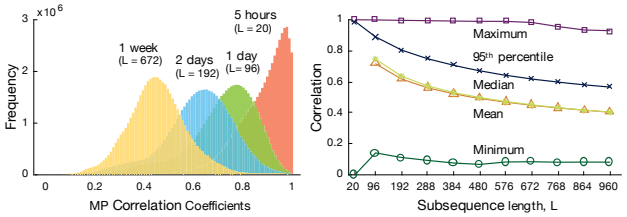



Fig. 4. (left) MP distance distributions for various lengths. (right) Percentile points for various lengths.

The number of subsequence lengths considered by the algorithm must be greater or equal to the number of different multi-way joins $k$, that the user wants. However, the algorithm can produce less than $k$ multi-way joins if the algorithm uses exactly $k$ lengths. Hence, a strategy has to be developed regarding which lengths to consider. We propose to use $k$ lengths equal spaced sampled between 10% to 30% of the length ($m$) of the time series at an equally spaced sample. If we fail to discover $k$ valid joins (e.g., trivial matches), we extend the range from 5% to 35% and sample more within this range until we reach $k$ different joins or exhaust all options.

### F. Trivial matches in the multi-length case

A well-known concept in pattern discovery for a single time series is trivial matches [18]. A trivial match occurs when a subsequence that begins at time $t$ best matches with adjacent subsequences starting at time $t+1$ or $t-1$. Since the subsequences share most of the values, these matches need to be avoided.

In our case, the pattern discovery is performed over multiple time series, and the patterns have multiple lengths. A trivial match occurs when a smaller pattern with length $p$ is discovered into a larger pattern with length $q$, where $p << q$. To avoid trivial joins, before adding a subsequence in a join, we ensure that the subsequence has less than 10% overlap between any subsequences in other joins. To find the best $k$ multi-way joins, we define the overlap score between two subsequences as $O = \frac{2s}{p+q}$, where $p$ and $q$ are the lengths of the subsequences and $s$ is the size of their shared region. Note that the overlap score between two subsequences from different time series is 0. Among all the $k$ multi-way joins found by our algorithm, we choose the one which has the minimum average pairwise overlap score.

## V. Experimental evaluation

To ensure the reproducibility of our solution, we created a website [19] where we made available all the codes, results, and data.

### A. Rival methods and data

In the experimental evaluation, we compare the patterns found by MultiPAL with those discovered by the Exact algorithm previously discussed in Section IV-B and by the algorithm of unsupervised shapelet discovery (**SUSh**) [20] and multivariate motif discovery (**mSTAMP**) [11]. Since we propose a novel sequential pattern mining task, the methods from the literature are not exactly multiple join algorithms. Thus, we adapt the available options to work for multiple joins. As SUSh and mSTAMP do not deal with patterns of different lengths, we consider the union of all results of multiple runs by varying the length parameter. For instance, we run the algorithms with lengths 35, 30, and 20 observations for the sinusoidal patterns dataset. To be fair with the competing methods, after extracting patterns for a length, we replace the patterns with white noise to discover new patterns of new lengths. Note that MultiPAL automatically selects these lengths to discover joins. Hence, it does not need this favor.

We consider six datasets: two datasets previously evaluated by SUSh (PAMAP and Trace), Horses [21], two datasets from our case studies (Power and Birds), which we will introduce in Section VI, and the synthetic dataset of sinusoidal patterns, previously introduced in Figure 2. Table I describes the datasets with the number of series, classes, and length.

TABLE I
DATASETS DESCRIPTION.

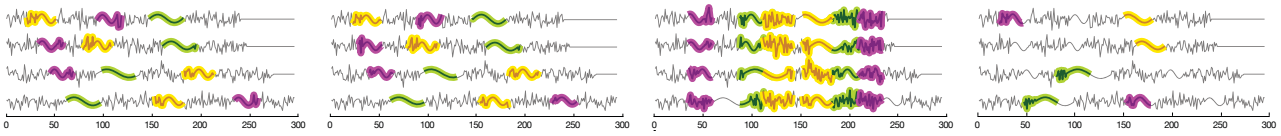
| | #Time series | Length | #Classes |
|---|---|---|---|
| Sinusoidal | 4 | 300 | 3 |
| Power | 85 | 35,040 | not available |
| Birds | 5 | 600 | 3 |
| Horses | 5 | 2,000 | 4 |
| Trace | 50 | 1,100 | 4 |
| PAMAP | 20 | 3,000 | 6 |

### B. Sanity check

First, we present a *sanity check* to illustrate that MultiPAL discovery similar patterns to the expensive Exact algorithm and that the current solutions from literature cannot discover multiple meaningful joins on multiple time series, leading to the need for a novel approach. For this, we consider the synthetic data previously shown in Figure 1 with the three sine patterns planted in random series.

Figure 5 shows the patterns discovered. mSTAMP, as a multivariate motif discovery method, identifies motifs that repeat over time. Thus, the algorithm identifies three motifs of three lengths, where all of them are synchronized in time. The algorithm missed at least four sinusoidal patterns and extracted white noises as motifs because the patterns are not synchronous in time. In contrast, SUSh, as an unsupervised shapelet discovery method, discovers only half of the sinusoidal patterns. SUSh clusters the time series based on the discovered shapelets. Since each time series contains multiple sinusoids of different lengths, clustering the time series using the same length shapelets hurts the performance of SUSh. A side-by-side comparison shows that repeated execution of the SUSh (unsupervised shapelet) algorithm is not equivalent to ground-up multiple joins (MultiPAL). Moreover, SUSh is a non-deterministic algorithm producing different results from different execution, while MultiPAL is deterministic.

MultiPAL, a multiple join method, identifies all twelve sinusoidal patterns of different lengths, repeating across all four time series. A small portion of noise is also captured for most patterns due to the similarities between these short noise segments. As we can note, all the patterns are equivalent to



(a) Our proposal: MultiPAL (b) Exact (c) mSTAMP (d) SUSh

Fig. 5. Multi-length patterns discovered over multiple time series by MultiPAL and algorithms from literature.

those discovered by the Exact method. However, MultiPAL is about five times faster.

### C. Quantitative evaluation

To analyze the quality of the patterns discovered by the methods, we quantitatively compare them using Adjusted Rand Index (ARI) [22] for datasets with ground truth labels. ARI is an external quality metric that essentially indicates what fraction of data was clustered correctly.

The ARI results and the number of patterns discovered by each algorithm are presented in Table II. This evaluation considers MultiPAL using two different correlation measures to compute the similarity between patterns: Pearson's Correlation Coefficient measured from Euclidean distance (ED) and Dynamic Time Warping (DTW) distance based correlation measured by $\frac{DTW\ distance}{2\times(path\ length)}$. We note MultiPAL with ED has shown the best results for all datasets, comparable to the Exact. Specifically, for Sinusoidal and Birds, we obtain a perfect clustering with MultiPAL (ED). For Trace and PAMAP, the Exact algorithm can not run due to memory constraints and impractical time caused by the larger number of time series and length. MultiPAL with DTW also shows an expensive execution time for these datasets. For Trace, MultiPAL (ED) discovers 200 patterns, while mSTAMP and SUSh discover 400 and 189, respectively. For PAMAP, MultiPAL (ED) discovers 120 patterns, while mSTAMP and SUSh discover 240 and 105 patterns, respectively. The results demonstrate that our method outperforms the rivals discovering a higher number of patterns than SUSh and fewer patterns with better ARI than mSTAMP for all datasets evaluated.

TABLE II
QUANTITATIVE COMPARISON USING ADJUSTED RAND INDEX AND THE NUMBER OF PATTERNS DISCOVERED.

| | MultiPAL (ED) | MultiPAL (DTW) | Exact | mSTAMP | SUSh |
|---|---|---|---|---|---|
| Sinusoidal | **1.0 (12)** | 0.78 (9) | **1.0 (12)** | 0.64 (24) | 0.73 (6) |
| Birds | **1.0 (15)** | 0.72 (15) | **1.0 (15)** | 0.61 (30) | 0.79 (8) |
| Horses | **0.82 (16)** | 0.80 (20) | **0.82 (16)** | 0.60 (32) | 0.62 (11) |
| Trace | **0.89 (200)** | – | – | 0.63 (400) | 0.68 (189) |
| PAMAP | **0.88 (120)** | – | – | 0.73 (240) | 0.81 (105) |

Table III shows the runtime of the methods. MultiPAL with DTW is the most expensive method, followed by SUSh and Exact. However, it must be noted that the high cost of MultiPAL (DTW) is mainly associated with the quadratic time complexity of DTW. Since the best ARI results of MultiPAL are observed using Euclidean distance, the employment of strategies to accelerate the DTW computation for multi-length pattern discovery is subject to future works. Although mSTAMP is the most efficient algorithm, the patterns discovered are not meaningful for multi-way time series join.

TABLE III
RUNTIME (IN SECONDS) COMPARISON.

| | MultiPAL (ED) | MultiPAL (DTW) | Exact | mSTAMP | SUSh |
|---|---|---|---|---|---|
| Sinusoidal | 0.60 | 14.92 | 2.99 | 0.19 | 22.93 |
| Birds | 1.52 | 260.14 | 29.48 | 0.55 | 71.35 |
| Horses | 1.64 | 14400.29 | 229.17 | 2.57 | 309.22 |
| Trace | 142.70 | – | – | 4.49 | 166.57 |
| PAMAP | 71.50 | – | – | 23.85 | 1226.99 |

### D. Scalability

Although our complexity analysis demonstrates that MultiPAL is orders of magnitude faster than the Exact solution, we compare the methods' execution times to show our results empirically using real-world data. The results are shown in Figure 6, where we can note the efficiency of MultiPAL. Since the Exact algorithm's memory requirement grows exponentially with the number of time series, the algorithm can not run for more than six time series.

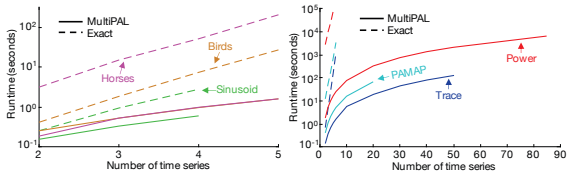



Fig. 6. Execution time varying the number of time series.

### E. Parameter sensitivity

There are two parameters to MultiPAL: $i$) the number of multi-way joins the user wants ($k$) and $ii$) the size of the memory to hold candidate joins ($M_t$). The former is a user choice, and the latter depends on the computing system's available memory. To test sensitivity, we manually pick two 30-days subsets of power consumption data (Section VI-A) considering the occurrence/absence of periodic patterns, to known: *periodic* and *aperiodic*.

We run MultiPAL on both of these datasets by varying $k$ and show the number of subsequence lengths we need to investigate. The results are shown in Figure 7(left). We observe that the number of lengths closely follows $k$ and increases drastically if the data does not have $k$ valid joins. In periodic data, the algorithm continues to find more joins until the whole dataset is explored.

We run MultiPAL for various memory sizes with a very large $k$. The results are shown in Figure 7(right). The number of joins discovered shows discrete upward changes as we add more memory. The algorithm needs more memory in aperiodic data to identify each additional join.

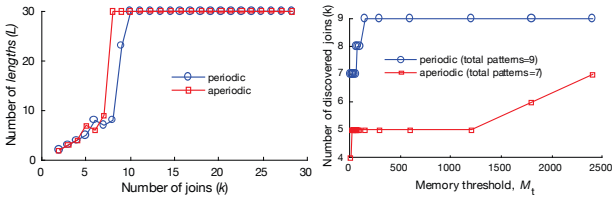
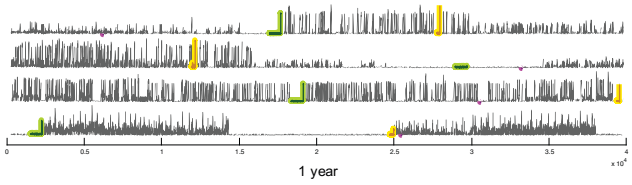



Fig. 7. Parameter sensitivity for periodic and aperiodic time series.

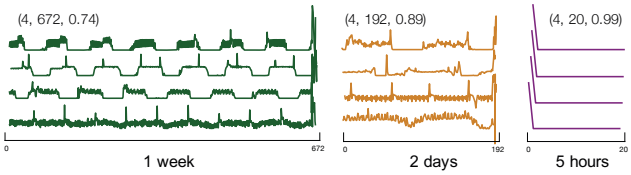
## VI. Case Studies

### A. Household power consumption

We have collected electric power consumption data from the Los Alamos Public Utility Department in New Mexico, USA. The dataset contains consumption data from 1,667 households, one sample every fifteen minutes. The dataset spans four years in duration.




(a) Power consumption of four customers over a year




(b) Multi-length patterns discovered by MultiPAL

Fig. 8. Analysis of power consumption.

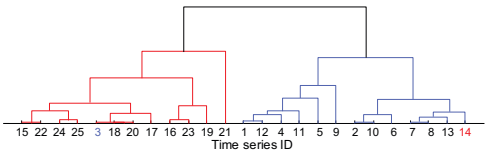



Fig. 9. Dendrogram for the Hierarchical clustering of 25 time series.

Figure 8(b) shows the three multi-way joins of three unique lengths. Each join is spanning 31 users in our dataset. We show four of these time series in Figure 8(a). The joins are all showing a sharp increase or decrease in consumption. However, the **1-week** join illustrates daily periodicity due to solar panels installed on the houses. The **2-days** join identifies similar periodic patterns containing houses with solar panels. The **5-hours** join identifies zero-power consumption patterns. Two of them are confirmed as outages, and the other two are periods when solar panels are sufficient to reduce consumption from the grid to zero. Note the progression from regular usage pattern to anomalous usage pattern as we decrease the join length, demonstrating the utility of multiple joins. All these results have been validated using satellite images on Google Maps to identify customers' houses with solar panels and querying the LADPU Twitter[1], where the company reports outages with date, time, and location.

*a) Clustering Utility:* Patterns obtained by MultiPAL are useful to cluster time series. We cluster 25 power consumption time series. The first thirteen time series are from the houses with solar panels and the remaining twelve do not have solar panels. We use the hierarchical clustering algorithm with the average linkage method. To compute the distance between two time series, we use the patterns obtained by MultiPAL from those time series as reference objects. We project each time series on a one dimensional plane by taking the average correlation over the ten nearest neighbors of each of the MultiPAL patterns in that time series. We then compute distances between any two time series as the absolute difference between their average correlations with MultiPAL patterns.

Figure 9 shows that the algorithm clusters the time series into two groups. The blue cluster contains 12 out of 13 time series from the solar panels group and one time series from the non-solar panels, yielding an Adjusted Rand Index (ARI) of **0.85**. Such projection based on discovered patterns is very common in the literature [6] [20], making *MultiPAL patterns useful in higher level data mining task.*

### B. Bioacoustic monitoring

An important task in biodiversity analysis is wildlife monitoring by acoustic sensors [23]. In general, these sensors generate a large volume of data in which manual analysis is tedious and time-consuming. Solutions such as MultiPAL can be an essential tool for analyzing bioacoustic data and finding behavioral patterns.

We built a dataset with audio records[2] containing bird calls of three species: $i$) Crex-crex, $ii$) Emberiza calandra, and $iii$) Aegolius funereus. We generated five 1-minute signals that contain calls of 20 seconds for each bird species in a shuffled order. Each call is generated by a different bird, even for those from the same species.

The original audio is high-dimensional (48kHz) with weak representational power. To reduce the dimensionality and obtain a strong representation, we converted it to Mel-frequency cepstrum coefficients (MFCC) space using a sliding window compressing 0.2 seconds, 50% of overlap, and keeping only the $2^{nd}$ coefficient [24]. Thus, we extracted 10 MFCC per second. In Figure 10, we illustrate the representational change for an Emberiza calandra call.

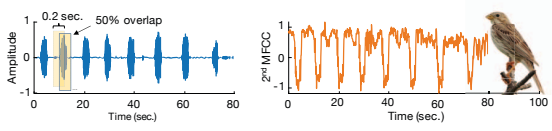
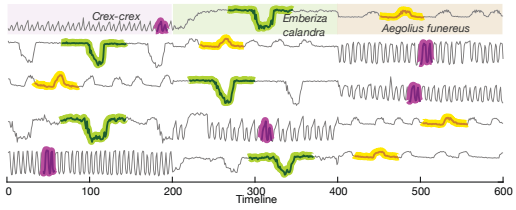



Fig. 10. Representational change of a bird call signal from time domain (left) to MFCC space (right).

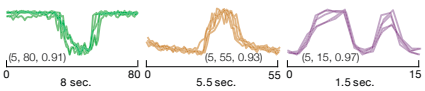
In Figure 11(a), we show the signals and highlight the multi-length patterns discovered by MultiPAL. For the first signal,

[1]https://twitter.com/ladpu
[2]http://xeno-canto.org/

we also indicate the class label for each 20-second segment according to the bird species. Our algorithm discovered 15 patterns distributed in groups of 8 seconds, 5.5 seconds, and 1.5 seconds duration. Given that MultiPAL discovered patterns with the same lengths for the same species and discovered a pattern for all the species, it scored an Adjusted Rand Index of 1 to separate data, which means a perfect clustering. To better visualize the quality of the patterns discovered, in Figure 11(b), we show the patterns separated by length and the multi-way joins $(N, L, C)$ provided by MultiPAL. These patterns among the calls with slight differences in vocalizations could be essential features to identify activities, such as when the birds are establishing the breeding territory, attracting females, or challenging intruding males.



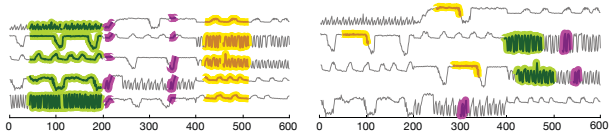

(a) Bird calls in MFCC space




(b) Multi-length patterns

Fig. 11. Multi-length patterns discovered by MultiPAL for Birds dataset.

The results of the rival methods are shown in Figure 12. mSTAMP identified 30 patterns distributed in three different lengths. All the patterns are synchronized over time, but dissimilar among them, leading to the worst ARI result (0.61). On the other hand, SUSh discovered eight similar patterns generating an ARI of 0.79. However, the method missed one Emberiza calandra pattern in the fourth signal and did not found any Aegolius funereus related pattern. We omitted the fifth signal since the SUSh did not discover any pattern for this time series.




(a) mSTAMP (b) SUSh

Fig. 12. Multi-length joins obtained by the rival methods SUSh and mSTAMP for Birds dataset.

## VII. Conclusion

This paper defines multiple joins for time series and describes a heuristic search algorithm for this novel task. The algorithm produces non-trivial results in comparison to iterative applications of known algorithms. Empirical case studies demonstrate the algorithm's usefulness in identifying characteristics patterns in real-world domains, including bioacoustics and electric power consumption monitoring.

**Acknowledgements:** This work was supported by the NSF Grant #OIA-1757207.

## References

[1] A. Mueen, H. Hamooni, and T. Estrada, "Time series join on subsequence correlation," in *ICDM*, 2014, pp. 450–459.
[2] C. M. Yeh, Y. Zhu, L. Ulanova, N. Begum, Y. Ding, H. A. Dau, D. F. Silva, A. Mueen, and E. Keogh, "Matrix profile i: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets," in *ICDM*, 2016, pp. 1317–1322.
[3] V. D. Vinh, N. P. Chau, and D. T. Anh, "An efficient method for time series join on subsequence correlation using longest common substring algorithm," in *ICCASA*, 2016, pp. 121–131.
[4] L. Chen, C. Rong, and J. Feng, "Time series join on most correlated subsequences using mapreduce," in *HPCC/SmartCity/DSS*, 2019, pp. 1366–1373.
[5] A. Mueen, "Time series motif discovery: dimensions and applications," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 4, no. 2, pp. 152–159, 2014.
[6] J. Zakaria, A. Mueen, and E. Keogh, "Clustering time series using unsupervised-shapelets," in *ICDM*, 2012, pp. 785–794.
[7] A. Mueen, "Enumeration of time series motifs of all lengths," in *ICDM*, 2013, pp. 547–556.
[8] Y. Gao and J. Lin, "Hime: discovering variable-length motifs in large-scale time series," *Knowledge and Information Systems*, vol. 61, no. 1, pp. 513–542, 2019.
[9] M. Linardi, Y. Zhu, T. Palpanas, and E. Keogh, "Matrix profile goes mad: variable-length motif and discord discovery in data series," *Data Mining and Knowledge Discovery*, pp. 1022–1071, 2020.
[10] D. Minnen, C. Isbell, I. Essa, and T. Starner, "Detecting subdimensional motifs: An efficient algorithm for generalized multivariate pattern discovery," in *ICDM 2007*, 2007, pp. 601–606.
[11] C. M. Yeh, N. Kavantzas, and E. Keogh, "Matrix profile vi: meaningful multidimensional motif discovery," in *ICDM*, 2017, pp. 565–574.
[12] Y. Gao and J. Lin, "Discovering subdimensional motifs of different lengths in large-scale multivariate time series," in *ICDM*, 2019, pp. 220–229.
[13] R. Perego, S. Orlando, and P. Palmerini, "Enhancing the apriori algorithm for frequent set counting," in *DaWak*, 2001, pp. 71–82.
[14] Z. Zimmerman, K. Kamgar, N. S. Senobari, B. Crites, G. Funning, P. Brisk, and E. Keogh, "Matrix profile xiv: Scaling time series motif discovery with gpus to break a quintillion pairwise comparisons a day and beyond," in *SoCC*, 2019, pp. 74–76.
[15] A. Mueen, Y. Zhu, M. Yeh, K. Kamgar, K. Viswanathan, C. K. Gupta, and E. Keogh, "The fastest similarity search algorithm for time series subsequences under euclidean distance," 2015, http://www.cs.unm.edu/~mueen/FastestSimilaritySearch.html.
[16] A. Mueen, E. Keogh, and N. Young, "Logical-shapelets: an expressive primitive for time series classification," in *KDD*, 2011, pp. 1154–1162.
[17] J. Zakaria, A. Mueen, E. Keogh, and N. Young, "Accelerating the discovery of unsupervised-shapelets," *DMKD*, vol. 30, no. 1, pp. 243–281, 2016.
[18] A. Mueen, E. Keogh, Q. Zhu, and S. Cash, "Exact discovery of time series motifs," in *SDM*, 2009, pp. 473–484.
[19] M. P. Mollah, V. M. A. Souza, and A. Mueen, "Supporting website," https://sites.google.com/view/multipal.
[20] L. Ulanova, N. Begum, and E. Keogh, "Scalable clustering of time series with u-shapelets," in *SDM*, 2015, pp. 900–908.
[21] J. W. Kamminga, L. M. Janßen, N. Meratnia, and P. J. Havinga, "Horsing around – a dataset comprising horse movement," *Data*, vol. 4, no. 4, p. 131, 2019.
[22] L. Hubert and P. Arabie, "Comparing partitions," *Journal of classification*, vol. 2, no. 1, pp. 193–218, 1985.
[23] R. Gibb, E. Browning, P. Glover-Kapfer, and K. E. Jones, "Emerging opportunities and challenges for passive acoustics in ecological assessment and monitoring," *Methods in Ecology and Evolution*, vol. 10, no. 2, pp. 169–185, 2019.
[24] D. F. Silva, V. M. A. Souza, and G. Batista, "A comparative study between mfcc and lsf coefficients in automatic recognition of isolated digits pronounced in portuguese and english," *Acta Scientiarum. Technology*, vol. 35, no. 4, pp. 621–628, 2013.