Matrix Profile Based kNN Search over Large Time Series

Tanmoy Mondal[1,2*], Reza Akbarinia[2] and Florent Masseglia[2]

[1*]ZENITH Team, INRIA & LIRMM, Montpellier, France.
[2]Mathematical & Electrical Engineering Department, IMT Atlantique, Brest, France.

*Corresponding author(s). E-mail(s):
tanmoy.mondal@imt-atlantique.fr;
Contributing authors: reza.akbarinia@inria.fr;
florent.masseglia@inria.fr;

Abstract

Matrix Profile (MP) has been recently proposed as a powerful technique for knowledge extraction from time series. Several algorithms have been proposed for computing it, *e.g.*, STAMP and STOMP. Currently, MP is computed based on 1NN search in all subsequences of the time series. In this paper[1], we claim that a kNN MP can be more useful than the 1NN MP for knowledge extraction, and propose an efficient technique to compute such a MP. We also propose a technique for parallel execution of the proposed algorithm by using multiple cores of an off-the-shelf computer. We evaluated the performance of our techniques by using multiple real datasets in our experiments. The results illustrate the superiority of our proposed kNN MP computation technique for knowledge discovery compared to the techniques for 1NN MP computations.

Keywords: Time series analysis, STAMP, STOMP, All-pairs-similarity search, Motifs and discord discovery, Outliers detection, Anomaly detection, Joins

1 Introduction

A time series is a series of data points ordered in time. As examples of time series, we can mention the height of ocean tides level captured every minute, the vibration of an aircraft engine captured every second, or the number of steps measured by a

smart watch day after day. Analyzing the time series can give us precious information about the underlying applications, *e.g.*, the anomalies in an aircraft engine.

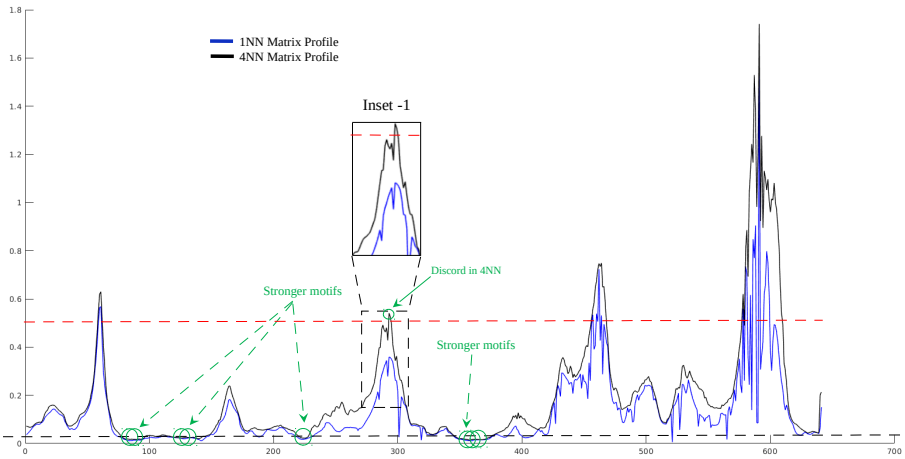


Figure 1: The 1NN and 4NN MP are plotted with different colors in which the motifs and discords are marked.

Matrix Profile (MP) [1] has been recently proposed as a powerful technique for time series analysis, *e.g.*, detecting motifs or anomalies. Efficient algorithms have been proposed for MP computation, *e.g.*, STAMP [2], STOMP [2] and SCRIMP++ [3]. The definition of MP in the literature is as follows [1]. Given a time series T and a subsequence length m, the MP returns for each subsequence included in T its distance to the most similar subsequence ($1NN$) in the time series. We call this type of MP a *1NN MP*. It is very useful for data analysis, *e.g.*, detecting the motifs (represented by low values), or anomalies (represented by high values).

Although 1NN MP has been shown useful for knowledge discovery, it has its own drawbacks and can miss some important motifs and anomalies (also called discords). Particularly, it does not allow to detect a cluster of discords, *e.g.*, two points that are similar to each other, but dissimilar to all other points. In addition, 1NN MP does not permit to distinguish a weak motif (*i.e.*, a point that has only one similar point in the series) from strong motifs (*i.e.*, those that have several similar points).

We believe that a more powerful MP based on kNN search is of high interest, where for each subsequence, its k^{th} nearest neighbor is used for generating the MP. We call it *kNN MP*. An example is depicted in Fig. 1 where *1NN* and *kNN* (*4NN* in this case) MPs are drawn for a time series of protein spectrum, representing protein rates measured in 10 different products (only certain portion of the complete MP is shown here). As seen in Fig.1, the *strong motifs* can be detected in *4NN* MP, whereas they are not either significantly visible or invisible in *1NN* MP. For example, the distance values in $1NN$ and $4NN$ MPs which are less than a user defined low threshold

value (marked as black dotted line at the bottom of the Fig. 1) are termed as *"strong motifs"*, because they have appeared in both the $1NN$ and $4NN$ MPs. This analogy intrinsically signifies that the subsequence has not only one close match (*i.e.*, *1NN* which may occur due to some noise or outlier elements) but multiple close matches, *i.e.*, *1NN*, *2NN*, *3NN* and *4NN*. This reasoning helps to increase the certitude of validating a subsequence as motif. An interesting case of discord is also shown in Fig. 1, where based on the user defined threshold (dotted red line in Fig. 1), the discord can be detected only in $4NN$ MP, while it was not detected in $1NN$ MP. There are situations where the distance in $1NN$ MP for an anomaly is quite low because the subsequence has one close match but the match can simply be another anomaly. But, the distance could be high in $4NN$ (or may be in $3NN$ or $5NN$) MP. A similar instance is shown in Inset 1 of Fig. 1 from a real data-set, which means that the third nearest neighbor (for the case $3NN$ MP) or fourth nearest neighbor (for the case of $4NN$ MP) of the subsequence is dissimilar. Hence we are not sure whether this subsequence should be considered as *motif* (based on it's $1NN$ and/or $2NN$) or as discord (based on it's $3NN$ and/or $4NN$). So to properly validate a subsequence as discord, the distance value should cross in $1NN$, $2NN$, $3NN$, ...kNN (where k is an user given parameter) MPs.

In this paper, we propose the kNN MP and illustrate its utility for knowledge discovery from time series. Our contributions are as follows:

1. We define the kNN MP, and propose a fast algorithm to calculate it in a time series.
2. We propose a technique for parallel execution of the proposed algorithm by using multiple cores of an off-the-shelf computer.
3. We evaluate the performance of our technique experimentally by using multiple real datasets. The experimental evaluation shows how qualitatively the *kNN* MP is useful for knowledge discovery compared to the *1NN* MP. The results also illustrate the efficiency of our solution for kNN MP computation[1].

The rest of the paper is organized as follows. In Section 2, we give the necessary background and definitions. In Section 3, we discuss the related work. The proposed algorithm is presented in Section 4 whereas the algorithm for parallel computation of the proposed *kNN* similarity search technique is presented in Section 5. In Section 6, an extensive experimental evaluation is reported. Finally, Section 7 concludes.

2 Problem Definition

In this section, we give the formal definition of kNN MP, and describe the problem we address.

Definition 1. ***Time series:*** *A time series T is a sequence of real-valued numbers $T = \langle t_1, \ldots, t_n \rangle$ where n is the length of T.*

A subsequence of a time series is defined as follows.

[1]The source code and the tested datasets are publicly available at: https://sites.google.com/view/knnmatrixprofile/home

Definition 2. ***Subsequence:*** *Let m be a given integer value such that $1 \leq m \leq n$. A subsequence T_i of a time series T is a continuous sequence of values in T of length m starting from position i. Formally, $T_i = \langle t_i, \ldots, t_{i+m-1} \rangle$ where $1 \leq i \leq n - m + 1$. We call i the start position of T_i.*

One of the primary goals of time series analysis is to perform time series subsequence matching. Given a positive real number τ (called *threshold*) and a time series T, two subsequences T_p (beginning at index p) and T_q (beginning at index q) of same length m, and a distance function $Dist(T_p, T_q)$ that measures the Euclidean distance between T_p and T_q. If $Dist(T_p, T_q) \leq \tau$, then T_q is called a *matching* subsequence of T_p.

Definition 3. ***Distance profile:*** The distance between a subsequence T_i with all other subsequences of the time series T gives a vector of distances, called distance profile of T_i.

The minimum value of this distance profile represents the closest match or $1NN$ and the top k minimum values of this vector represent the kNN matches of the subsequence T_i. A *MP* can be defined as a vector of nearest neighbor ($1NN$) distances of all the subsequences of time series T.

Definition 4. ***kNN MP:*** *The kNN MP of a time series T is a vector $P = \langle p_1, \ldots, p_{n-m+1} \rangle$ such that p_i is the distance of the subsequence T_i to its k^{th} nearest neighbor among the subsequences of T.*

The *kNN MP index* is a vector $I = \langle s_1, \ldots, s_{n-m+1} \rangle$ such that s_i is the index of the k^{th} nearest neighbor of the subsequence T_i in the time series T.

Definition 5. ***Motif:*** *A motif pair is defined as a pair of subsequences $\langle T_i, T_j \rangle$ whose distance is less than a user defined threshold (i.e., $Dist(T_i, T_j) \leq \tau$), and their starting positions are at least w elements apart ($|i - j| \geq w$), where w is a user defined threshold which tells that two subsequences in a pair should be w elements apart. If $|i - j| < w$, then the motif pair $\langle T_i, T_j \rangle$ is called a trivial match.*

From the definition of *1NN* motif, we can define the *kNN* motif as follows.

Definition 6. ***kNN Motif:*** *A subsequence T_i is a* kNN motif *if its distance to its k^{th} nearest neighbor (say T_j^k) is less than the user defined threshold (i.e. $Dist(T_i, T_j^k) \leq \tau$) and their starting positions are at least w elements apart.*

Traditionally, a discord (or anomaly) is defined as a subsequence (say T_i) whose distance from all other subsequences of the time series is high that means it has no close match. Hence the distance between T_i and it's nearest neighbor is high. Let us now define the kNN discord.

Definition 7. ***kNN Discord:*** *Let T be a time series, and ω a high distance threshold given by the user. A subsequence $T_i \in T$ is a kNN discord, if the distance of T_i to its k^{th} nearest neighbor is higher than ω.*

A $1NN$ discord is a subsequence whose distance to its nearest neighbor is higher than ω.

Let us now define the problem which we address in this paper. Given a number k, a time series T, and a subsequence length m, our goal is to efficiently compute the kNN MP.

3 Related Work

MP is an efficient solution to the problem of *similarity join*, which can be defined as: given a set of data objects (for our case subsequences), retrieve the nearest neighbors for each object. The solution to this problem can be useful for motif and anomaly discovery from time series in many application domains such as bioinformatics [4], speech processing [5], Seismology [6], etc. Similarity join can be categorized into two principal categories, *1NN similarity join* and *kNN similarity join*. With *kNN similarity join*, for each given object the k nearest neighbors are returned, where k is a positive number given by the user. The *1NN similarity join* is a special case of *kNN similarity join* with $k = 1$. To the best of our knowledge, the MP algorithms in the literature only take into account the 1NN category.

The authors in [7] propose an approach to optimize the calculation of *Euclidean distance* between all subsequences. The idea is to interleave the early abandoning calculations of *Euclidean distance* with the concept of online *z-normalization*. In [8], the authors propose an algorithm based on intelligent caching, reusing computations and the pruning of the search space. By reusing computations of z-normalized distances for overlapping subsequences, the authors have highly saved the computation time and reduced the search space significantly.

In [9] [8], Mueen et al. propose MASS, an efficient algorithm for similarity search in time series. It exploits the consecutive subsequence overlapping property to calculate *Z normalized* distance by *Fast Fourier Transform (FFT)* based convolution. Thanks to the use of *FFT*, in recent years the MASS algorithm has emerged as a significant contribution in subsequence similarity search for many similarity based pattern matching problems such as motif and discord discovery, nearest neighbor matching, etc. [10], [11], [12], [13].

Yeh et al. [2] proposed *MP*, an efficient technique for *similarity join* in time series [14]. The authors use the convolution property of *FFT* and *Inverse FFT* for the fast calculation of MP by an algorithm called STAMP. Furthermore, an incremental algorithm, called STOMP, is adapted for distance computation of overlapping sequential subsequences in which they use *MASS* algorithm for time series similarity search by computing *z-normalized* euclidean distance between the subsequences. However, the techniques proposed in [2] are mainly designed for *1NN similarity join*, and it cannot be trivially adapted for *kNN similarity join*.

In [3], the authors introduce an anytime algorithm, named as *SCRIMP++* by combining the best features of *STAMP* and *STOMP* for fast MP computation. But this technique is also designed for *1NN* MP.

The state-of-the-art techniques for calculating MP (*e.g. STAMP* and *STOMP*) can only compute $1NN$ MP (because these algorithms overwrites the MP values in each iteration) by using *MASS* algorithm (which is capable to give $1NN$ and kNN of a sub-sequence). To the best of our knowledge, in the literature there is no efficient solution for computing kNN MP (by using the same *MASS* algorithm). As illustrated in Section 1, this type of MP can be very useful for knowledge discovery from time series. In this paper, we propose an efficient solution for computing it.

4 kNN MP

Here, we have discussed the proposed technique for kNN MP computation of a time series T in a sequential computing environment.

4.1 MP Algorithm

Let's consider T be a big time series (which may be obtained by concatenating several small individual time series) and the goal is to find the k closest matches of all the subsequences $T[i]$ in T, where the matches should come from the same time series T and these matches should be separated by w elements from the subsequence in question *i.e.*, $T[i]$. Our algorithm for computing kNN MP of the time series T is inspired by the STOMP algorithm [15]. The pseudo-code of our algorithm is depicted in Algorithm 1. It basically computes the distance of each subsequence of $T[i]$ with all the remaining subsequences of T iteratively, and each time keeps only the kNN matches of each subsequence.

The fast computation of mean and standard deviation (STD) of the time series T is performed in Line 10 of Algorithm 1. Then, the *MASS* algorithm is applied in Line 12 to compute the dot product (QT) (distance D^{ignore} is ignored) between the first subsequence $(subSeq_1)$ and other subsequences of time series T. As the arguments of *MASS*, 1^{st} subsequence $(subSeq_1)$, mean $(\mu_T[1])$ and STD $(\sigma_T[1])$ of $subSeq_1$, complete time series T, mean (μ_T) and STD (σ_T) vector (*i.e.*, mean and STD of all the subsequences) of T are passed.

Then, the algorithm loops through all the subsequences of T (see Line 16) and iteratively obtains each subsequence $(cutSubSeq)$ in Line 17. Only for the first subsequence (when $i = 1$), it applies *MASS* algorithm to compute the dot product (QT) and distance vector $(Dist_{cutSubSeq})$ between first target subsequence $(cutSubSeq)$ and the remaining subsequences of time series T. Contrary to Line 14, here we pass as arguments the subsequence $(cutSubSeq)$, mean $(\mu_T[i])$, STD $(\sigma_T[i])$ of $cutSubSeq$ and entire time series (T), mean (μ_T) and STD (σ_T) of T. For $i > 1$ on-wards, the distance of target subsequence $(cutSubSeq)$ with all the remaining subsequences of T is incrementally calculated using *IndependentSTOMP* algorithm. The following arguments are passed in *IndependentSTOMP* for the distance calculation: each subsequences $(cutSubSeq)$, the mean $(\mu_T[i])$ and STD $(\sigma_T[i])$ of T, the dot product value $(QT_{initial}[i])$ of very first subsequence $(subSeq_1)$ and i^{th} subsequences $(T[i$

to $(i + m - 1)]$), already computed dot product vector (QT), mean (μ_T) and STD (σ_T) of T.

Algorithm 1: kNN-TimeSeriesJoin($\mathfrak{D}^T$, m)

Input: The target time series data base $(\mathfrak{D}^T)$
Output: A MP (P_T) and associated MP index (I_T)

$n_{\mathfrak{D}^T} \leftarrow length(\mathfrak{D}^T)$ *// count the total number of time series in the data base* $\mathfrak{D}^T$
$Len^1 \leftarrow lenth(\mathfrak{D}^T[1])$ *// get the length of first time series from database* $\mathfrak{D}^T$
$T \leftarrow \mathfrak{D}^T[1]$ *// create a new vector* T *and initialize it by copying the first time series* $\mathfrak{D}^T[1]$ *in it*
$Info_{conCat} \leftarrow [1, Len^1, `file1.csv']$ *// when an individual time series is merged in* T*, keep the indexes at where it starts and ends in concatenated time series* T

for $iSeries \leftarrow 2$ ***to*** $n_{\mathfrak{D}^T}$ **do**
  $T \leftarrow [T, \mathfrak{D}^T[iSeries]]$ *// keep concatenating individual time series from the data base* $\mathfrak{D}^T$
  $Info_{conCat} \leftarrow [startIdx, endIdx, fileName]$ *// store the start, end indexes and the file name after concatenating an individual time series in* T

$n_{conCat} \leftarrow length(T)$ *// get the length of concatenated time series*
$Idxs_{conCat} \leftarrow n_{conCat} - m + 1$ *// get the total number of subsequences in* T

$[\mu_T, \sigma_T] \leftarrow ComputeMeanStd(T)$ *// see Equation 1 in Appendix: II*
$subSeq_1 \leftarrow T[1 \text{ to } (1 + m - 1)]$ *// get the* 1^{st} *subsequence from* Q
$[QT, D^{ignore}] \leftarrow MASS(subSeq_1, \mu_T[1], \sigma_T[1], T, \mu_T, \sigma_T)$ *// apply* $MASS$ *with arguments as* 1^{st} *subsequence of* T *i.e.* $subSeq_1$ *and remaining subsequences of time series* T*. See Algorithm 7 in Appendix: III*

$QT_{initial} \leftarrow QT$ *// keeping a copy of the very first dot product*
$P_T \leftarrow$ Initialize this $1D$ vector with inf
$I_T \leftarrow$ Initialize this $1D$ vector with zeros

for $i \leftarrow 1$ ***to*** $Idxs_{conCat}$ **do**
  $cutSubSeq \leftarrow T[i \text{ to } (i + m - 1)]$ *// get target subsequence by chopping* T *from index* i *to* $(i + m - 1)$
  if $i == 1$ **then**
    $[QT, Dist_{cutTarget}] \leftarrow MASS(cutSubSeq, \mu_T[i], \sigma_T[i], T, \mu_T, \sigma_T)$ *// apply* $MASS$ *with arguments as* 1^{st} *subsequence of* T *and complete time series* T*.*
  else
    $[QT, Dist_{cutSubSeq}] \leftarrow IndependentSTOMP(cutSubSeq, \mu_T[i], \sigma_T[i], QT_{initial}[i], T, QT, \mu_T, \sigma_T)$ *// repetitively calculate distance between a subsequence and all the remaining subsequences in* T*. See Algorithm 9 in Appendix: V*
  $P_T \leftarrow computeElementwiseMin(Dist_{cutSubSeq}, P_T)$ *// perform element-wise minimum value comparison of* P_T *and* $Dist_{cutSubSeq}$ *then iteratively maintain the minimum values in* P_T
  $I_T \leftarrow i$ *// update* I_T *at the indexes where element-wise minimum operation replaces the previously stored value in* P_T *with the new value from* $Dist_{cutSubSeq}$

return P_T and I_T *// return the* P_T *and* I_T *array*

Then as usual the MP array (P_T) and index profile array (I_T) are updated based on the distance of each subsequence $(cutSubSeq)$ of T (see Line $24-25$). Finally, these two vectors (P_T) and (I_T) are returned as the results of this algorithm.

4.2 Managing kNN subsequences

Most of the recent methods in the literature (*e.g.*, *STAMP* and *STOMP* algorithms [2]) are designed to find the best match ($1NN$-join) corresponding to each subsequence. In this section, we propose three techniques to find and manage the kNN closest matches for each subsequences: 1) Sort based; 2) Maximum based; 3) Heap based.

Algorithm 2: SORTING-BASED-KNN ($\mathfrak{D}^T$, m)

Input: The target time series data base T, user given k nearest neighbors kNN

Output: A MP P_T and associated MP index I_T

....

....

14 $P_T \leftarrow$ Initialize this $2D$ vector of size $\{(kNN+1) \times Idx_T\}$ with inf

15 $I_T \leftarrow$ Initialize this $2D$ vector of size $\{(kNN+1) \times Idx_T\}$ with $zeros$

16 **for** $i \leftarrow 2$ ***to*** $Idxs_{conCat}$ **do**

17 $cutSubSeq \leftarrow ...$

....

....

22 **if** $i <= kNN$ **then**

23 **for** $p \leftarrow 1$ ***to*** Idx_T **do**

24 $P_T[i,p] \leftarrow Dist_{cutTarget}[p]$

25 $I_T[i,p] \leftarrow i$

else

26 **for** $p \leftarrow 1$ ***to*** Idx_T **do**

27 $P_T[(kNN+1),p] \leftarrow Dist_{cutTarget}[p]$

28 $I_T[(kNN+1),p] \leftarrow i$

29 $[sortVals, sortIdxs] \leftarrow Sort(P_T[1 \text{ to } (kNN+1),p])$

30 **for** $t \leftarrow 1$ ***to*** kNN **do**

31 $P_T[t,p] \leftarrow sortVals[t,p]$

32 $I_T[t,p] \leftarrow I_T[sortIdxs[t],p]$

33 $P_T^{final} \leftarrow P_T[1 \text{ to } kNN, 1 \text{ to } Idx_T]$ *// perform element-wise minimum of value comparison of P_T and $Dist_{cutSubSeq}$ then keep the minimum values in P_T*

34 $I_T^{final} \leftarrow I_T[1 \text{ to } kNN, 1 \text{ to } Idx_T]$ *// update I_T at indexes where element-wise minimum operation replaces the previously stored value in P_T with the new value from $Dist_{cutSubSeq}$*

return P_T^{final} and I_T^{final} *// return the P_T^{final} and I_T^{final} array*

4.2.1 Sort based kNN search

Given a time series T, we need to keep updated the list of the k *nearest neighbors* of each subsequence T_i, when its distance with a subsequence of T is calculated. The idea of the sort based approach is to create a list containing the distance of the current kNN matches and the new computed distance, sort the list and take the first k distances and their corresponding subsequences.

The pseudo-code of the sort based approach is mentioned in Algorithm 2 which is the same as Algorithm 1 until Line 13 (so we avoid to mention it again). In Line $14-15$, two $2D$ arrays, named as : P_T and I_T are created for storing kNN. Both of these arrays are of size $\{(kNN+1) \times Idx_T\}$. $(kNN+1)$ number of rows are needed to keep k nearest neighbors, and the $(k+1)^{th}$ row is needed to temporarily hold newly calculated distance. In Line 16, we loop through all the subsequences of time series (T) and in Line 17, each subsequence is chopped as usual (same as Algorithm 1)). After that until Line 21, we perform the same operations as in Line $18-21$ of Algorithm 1. The initial kNN number of distances and index profiles are simply stored in P_T and I_T arrays (see Line $22-25$). From $(kNN+1)^{th}$ subsequence on-wards (the *else* portion in Line $26-32$), the newly calculated distance profile is saved at $k+1^{th}$ row (Line 27) and consequently the target subsequence index **i** is stored in $k+1^{th}$ row (Line 28). Then the distances stored from index 1 to $(kNN+1)$ are sorted in ascending order and from this sorting operation, we will get the sorted distance values $(sortVals)$ and their corresponding indexes $(sortIdxs)$ (see Line 29). After that the top kNN sorted values are updated in P_T matrix (Line 31) and corresponding stored indexes are also updated (Line 32) with the help of sorted indexes $(sortIdxs)$. This process is repeated iteratively for all the subsequences (Line $16-32$) and finally the MP P_T and the MP index I_T from the index 1 to kNN are returned (as P_T^{final} and I_T^{final}) as the output of this algorithm.

The average time complexity of sorting k elements is $\mathcal{O}(k \log k)$ and we need to perform this O$((n-m)^2)$ times, where n is the length of the time series T, and m the subsequence length. Hence, the total complexity is O$((n-m)^2 \times k \log k)$. To improve the computational time, in the next subsection we propose to replace *sorting* by finding *maximum* of top k distance values for each subsequences.

4.2.2 Maximum based kNN search

Here, instead of *sorting*, we calculate the *maximum* of every top k distance values and then compare these maximum value with the newly computed distance. The time complexity of finding maximum is $\mathcal{O}(k)$ which is already less than the time complexity of the sort based algorithm, *i.e.*, $\mathcal{O}(k \log k)$. As the objective is to keep the smallest k distances for each subsequence, which can be simply fulfilled if we can repetitively take away the maximum value from top $(kNN+1)$ values. The pseudo code of this approach is mentioned in Algorithm 3. The *Else* portion (Line $26-32$) of Algorithm 2, needs to be replaced by the pseudo code of Algorithm 3 (Line $26-32$). As usual, the newly calculated distance profile and subsequence index are stored at $(kNN+1)^{th}$ row of P_T and I_T matrix (Line $27-28$). Then, we find the maximum value of top kNN elements and corresponding index from P_T matrix (Line 29). Now

this maximum value is compared with the newly arrived value which is temporarily kept at $(kNN+1)^{th}$ index (Line 30). If the newly arrived value is less than the existing maximum value, then the old maximum value is replaced by the new value ($P_T[maxIdx, p]$) (Line 31) and it's index is stored in I_T matrix (Line 32). This process is repeated for all the subsequences and finally the results P_T^{final} and I_T^{final} are returned as output of the algorithm.

Algorithm 3: MAX-BASED-KNN ($\mathfrak{D}^T$, m)

```
....
....
else
26    for p ← 1 to Idx_T do
27        P_T[(kNN+1), p] ← Dist_cutTarget[p]
28        I_T[(kNN+1), p] ← i
29        [maxVal, maxIdx] ← FindMax(P_T[1 to kNN, p])
30        if (P_T[(kNN+1), p] < maxVal) then
31            P_T[maxIdx, p] ← P_T[(kNN+1), p]
32            I_T[maxIdx, p] ← I_T[(kNN+1), p]
33  P_T^final ← P_T[1 to kNN, 1 to Idx_T]
34  I_T^final ← I_T[1 to kNN, 1 to Idx_T]
....
return P_T and I_T // return the P_T and I_T array
```

4.2.3 Heap based kNN search

Finding the maximum of a vector in classical manner has a time complexity of $\mathcal{O}(k)$. Here, we propose to find the maximum of a vector by using heap based priority queue whose time complexity is $\mathcal{O}\log(k)$. At first, we need to organize the elements into a heap structure so that the first element in the heap will contain the maximum value of the array. A new element is initially inserted at the end of heap and then again a heap based priority queue restructuring operation is performed so that we can obtain the maximum value at the top of array, having worst case time complexity ($\mathcal{O}\log(k)$).

The pseudo-code of the heap based approach is shown in Algorithm 4. Until Line 26, the algorithm remains the same as Algorithm 2. The *Else* portion (Line $26-32$) of Algorithm 2, needs to be replaced by the pseudo code of Algorithm 4 (Line $26-38$). In Line 29, we verify whether $i == (kNN+1)$, *i.e.* when we are handling $(kNN+1)^{th}$ subsequence of target, the elements of P_T array are organized for the first time in the structure of heap based priority queue. So in Line 30, we loop through each subsequence and for each of such subsequences, we organize the elements (kNN elements) of each column in a heap based priority queue. The heap structure is updated in P_T (Line 31). Thanks to this operation, we will have the maximum value at the first row of P_T matrix. Now, again we loop through all subsequences (Line 33) and compare the maximum value with the newly arrived value at $(k+1)^{th}$ index. If this newly arrived value is less than the existing maximum value (stored in first row) then we replace the value/s in the first row with the value/s at

Algorithm 4: HEAPMAX-BASED-KNN ($\mathfrak{D}^T$, m)

```
....
....
else
26    for p ← 1 to Idx_T do
27        P_T[(kNN + 1), p] ← Dist_cutTarget[p]
28        I_T[(kNN + 1), p] ← i
29    if i == (kNN + 1) then
30        for p ← 1 to Idx_T do
31            [P_T[1 to kNN, p], heapSortIdxs] ← BuildMaxHeap(P_T[1 to
                kNN, p], kNN)
32            I_T[1 to kNN, p] = I_T[heapSortIdxs[1 to kNN], p]
33    for p ← 1 to Idx_T do
34        if (P_T[(kNN + 1), p] < P_T[(1), p]) then
35            P_T[1, p] ← P_T[(kNN + 1), p]
36            I_T[1, p] ← I_T[(kNN + 1), p]
37            [P_T[1 to kNN, p], heapSortIdxs] ← BuildMaxHeap(P_T[1 to
                kNN, p], kNN)
38            I_T[1 to kNN, p] = I_T[heapSortIdxs[1 to kNN], p]
39  P_T^final ← P_T[1 to kNN, 1 to Idx_T]
40  I_T^final ← I_T[1 to kNN, 1 to Idx_T]
....
return P_T and I_T // return the P_T and I_T array
```

$(k+1)^{th}$ row (Line 35) and the corresponding index/s are also updated (Line 36). Then, we reapply the heap based priority queue technique to put the maximum value out of top kNN values at the first row (corresponding indexes are also updated, see Line $37-38$). This process is repeated for all the subsequences (see Line 16 in Algorithm 2) in order to produce the final kNN distances and corresponding indexes in P_T^{final} and I_T^{final} matrices (Line $39-40$).

5 Multi-core based parallel computing

Here we propose an approach to perform the parallel computation of kNN MP by exploiting multiple cores. Let $\mathfrak{n}$ be the total number of available cores, then the idea is to divide the total number of individual time series *i.e.* $n_{\mathfrak{D}^T}$ into $\mathfrak{g}$ number of groups (*i.e.*, $\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_{\mathfrak{g}}$) and to give each group into one individual core; where $\mathfrak{g} = \mathfrak{n}$. An example of kNN MP computation by using multiple cores is depicted in Fig. 2. As seen, the time series T is equally divided into 6 groups, and each group is processed by a separate core. The pseudo code of this technique is shown in Algorithm 5. Lines $1-10$ are the same as in Algorithm 1, where the process of concatenating $n_{\mathfrak{D}^T}$ number of small time series of different lengths i.e. $\{\mathfrak{t}_1, \mathfrak{t}_2, \mathfrak{t}_3, ..., \mathfrak{t}_{n_{\mathfrak{D}^T}}\} \in \mathfrak{D}^T$ from

the time series database $\mathfrak{D}^T$ is presented. By concatenating all the time series in $\mathfrak{D}^T$, we can obtain a big time series T (see line 1-7 in Algorithm 1)[2].

In Line 11, we obtain the number of available cores. If the remainder after division between the total number of individual time series, *i.e.*, $n_{\mathfrak{D}^T}$ and total number of available cores *i.e.* n_{cores} is zero then the variable $\mathcal{Y}$ (Line 16) will simply hold the indexes of first and last individual time series of a group (each group will contain $\mathfrak{g}$ number of time series). Otherwise, if the remainder is more than zero then the number of individual time series in each group is calculated by subtracting $\mathfrak{r}$ from $n_{\mathfrak{D}^T}$ and then dividing it by n_{cores} (Line 19). The indexes of first and last individual time series of all the groups except last group are stored in $\mathcal{Y}$ (Line 21). Whereas, the remaining $\mathfrak{r}$ number of time series, belongs to the last group are stored at the last cell of $\mathcal{Y}$ in Line 22.

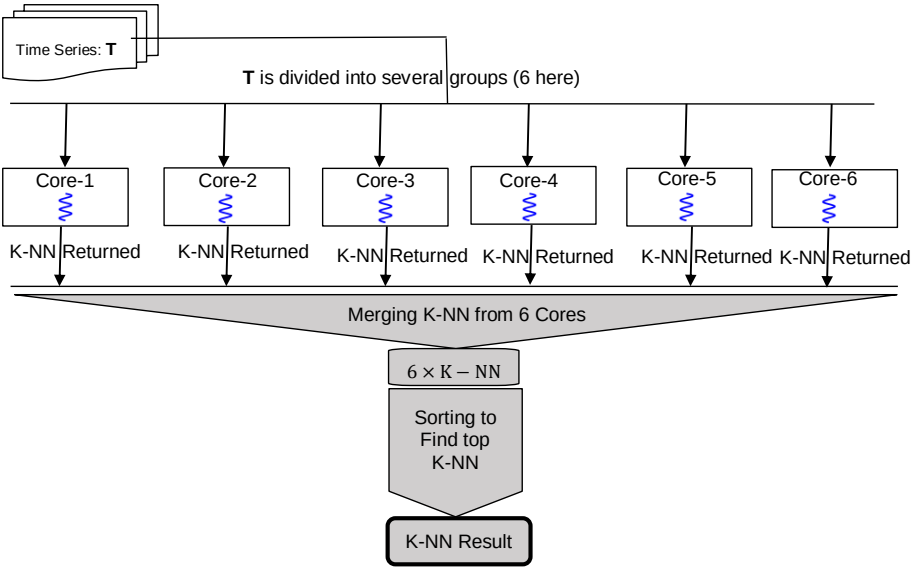


Figure 2: The proposed architecture of multi core based parallel processing.

Figure 3: The start and end indexes of each portions/parts of time series T, after dividing it into $\mathfrak{g}$ (= 6 here) number of groups.

By using the variable $Info_{conCat}$ (which keeps the indexes and length of each individual time series) and the variable $\mathcal{Y}$ (which keeps the indexes of first and last

[2]this case is only applicable when we don't have a single big time series and only several small time series are available in the database. Hence, a big time series is formed by concatenating these small time series

individual time series of all the groups), we can obtain start and end indexes of each parts/portions of time series T, which is divided into $\mathfrak{g}$ parts (Line $25-26$). Such an illustration is shown in Fig. 3, where the start (St) and end (Ed) indexes of each parts are visually depicted. In every core, the local kNN matches are calculated in parallel for all the subsequence of T by using the *"kNN_Join_Parallel ()"* function (Line 27). The idea is to calculate kNN matches of all the subsequences of T, where the matches come from one part/portion of the time series T, handled by an individual core. Then, the partial kNN MPs from different cores are merged and the global MP is produced. The MP P_T^{core} and index profile I_T^{core}, obtained from each core are saved in P_T^{All} and I_T^{All} respectively (Line 28). These matrices are concatenated in P_T^{concat} and I_T^{concat} respectively (Line $31-32$). The concatenated MP P_T^{concat} is then sorted column wise to obtain the sorted MP P_{sort} and the obtained sorted indexes $indx_{sort}$ are then used to obtain the sorted index profile I_{sort} (Line $33-34$). Finally, the top kNN rows of P_{sort} and I_{sort} are returned in P_T^{Final} and I_T^{Final} respectively. The function *"kNN_Join_Parallel ()"* is exactly the same as the Line $11-24$ of Algorithm 1 except[3] here the 1^{st} subsequence is obtained by using the $\mathfrak{st}$ location (Line $11-12$) and the loop in Line 16 that runs from $\mathfrak{st}$ to $\mathfrak{ed}+m-1$. In the Supplementary Materials SM: I, we have mentioned two special cases to handle the kNN matches where in the first case, we discuss about handling the matches, coming from two independent and individual time series. Whereas, in the second case, we talk about handling the matches, coming from two border files, treated by two separate cores of a computer. The experimental results, mentioned in Section 6.2.1 and 6.2.2 are obtained by employing these two special cases.

6 Experimental Evaluation

We evaluate the performance of our proposed kNN MP technique and illustrate its utility for motif and anomaly detection from time series.

6.1 Setup

In our experiments, we have used several datasets. The first dataset is *chemometric data*, representing protein rate measured on 10 different products. The second dataset is *accelerometer data* obtained by attaching accelerometer at the neck of 13 sheep. The third dataset is *seismic data* and the fourth dataset is a synthetic *random walk* dataset. We have also used some datasets from the UCR time series data mining archive [16] (see the list in Table 1). All the experiments are designed in a manner such that they can be easily reproducible [4]. The experiments done to evaluate the utility of kNN MP are performed on a classic off-the-shelf computer, having *Intel(R) Core(TM) i7-8850H CPU @ 2.60 GHz* processor with 32 GB RAM. The multi-core experiments are performed on a computational server having 36 physical cores.

[3] that's why we kept the same line number here to facilitate the understanding

[4] We have shared the code, datasets and instructions in : https://github.com/tanmayGIT/kNN_Matrix_Profile

Algorithm 5: KNN-PARALLEL($\mathfrak{D}^T$, m)

Input: The target time series data base ($\mathfrak{D}^T$)
Output: A MP (P_T) and associated MP index (I_T)

$n_{\mathfrak{D}^T} \leftarrow length(\mathfrak{D}^T)$ *//count the total number of time series in the data base* $\mathfrak{D}^T$
.....
$Info_{conCat} \leftarrow [1, Len^1, `file1.csv']$
.....
$[\mu_T, \sigma_T] \leftarrow ComputeMeanStd(T)$ *// see Equation 1 in Appendix: II*
$n_{cores} \leftarrow cluster.numWorkers$ *// getting the number of workers/cores available*
$\mathcal{Y} \leftarrow (n_{cores} \times 2)\ array$ *// these are 2D matrix, initialized with zeros*
if $(n_{\mathfrak{D}^T}\ \%\ n_{cores} == 0)$ **then**
  $\mathfrak{s} \leftarrow 1;\ \mathfrak{g} \leftarrow n_{\mathfrak{D}^T}/n_{cores}$ *// number of individual time series in each group is calculated in* $\mathfrak{g}$
  for $iClus \leftarrow 1$ ***to*** n_{cores} **do**
    $\mathcal{Y}[iClus][1] \leftarrow \mathfrak{s};\ \mathcal{Y}[iClus][2] \leftarrow \mathfrak{s} + \mathfrak{g} - 1;\ \mathfrak{s} = \mathfrak{s} + \mathfrak{g}$ *// the indexes of first and last individual time series is stored in* $\mathcal{Y}$
else
  if $(n_{\mathfrak{D}^T}\ \%\ n_{cores} > 0)$ **then**
    $\mathfrak{s} \leftarrow 1;\ \mathfrak{r} \leftarrow n_{\mathfrak{D}^T}/n_{cores};\ \mathfrak{g} \leftarrow (n_{\mathfrak{D}^T} - \mathfrak{r})/n_{cores}$
    for $iClus \leftarrow 1$ ***to*** $n_{cores} - 1$ **do**
      $\mathcal{Y}[iClus][1] \leftarrow \mathfrak{s};\ \mathcal{Y}[iClus][2] \leftarrow \mathfrak{s} + \mathfrak{g} - 1;\ \mathfrak{s} = \mathfrak{s} + \mathfrak{g}$
    $\mathcal{Y}[n_{cores}][1] \leftarrow \mathfrak{s};\ \mathcal{Y}[n_{cores}][2] \leftarrow n_{\mathfrak{D}^T};$ *// the indexes of start and last individual time series of last group is saved in the last cell of* $\mathcal{Y}$
$P_T^{All}, I_T^{All} \leftarrow (n_{cores} \times 1)\ array$ *// these are 1D vectors whose each cell contains a 2D matrix*
/* `// following for loop is run in parallel i.e. the contents inside the loop are given to each core` */
for $iCore \leftarrow 1$ ***to*** n_{cores} **do**
  $\mathcal{ST} = Info_{conCat}[\ \mathcal{Y}[iCore][1]\ ]$ *// get the starting index of the part/portion of full time series, handled by this core*
  $\mathcal{ED} = Info_{conCat}[\ \mathcal{Y}[iCore][2]\ ]$ *// get the end index of the part/portion of full time series, handled by this core*
  $[P_T^{core}, I_T^{core}] \leftarrow kNN\_Join\_Parallel(T,\ \mu_T,\ \sigma_T,\ \mathcal{ST},\ \mathcal{ED},\ m)$
  $P_T^{All}[iCore] \leftarrow P_T^{core};\ I_T^{All}[iCore] \leftarrow I_T^{core}$
$P_T^{concat} \leftarrow P_T^{All}[1];\ I_T^{concat} \leftarrow I_T^{All}[1]$ *// initialized with the first matrices i.e.* $P_T^{All}[1]$ *and* $I_T^{All}[1]$
$nLen \leftarrow n_{cores} \times kNN$
for $i \leftarrow 2$ ***to*** n_{cores} **do**
  $P_T^{concat} \leftarrow [P_T^{concat}, P_T^{All}[i]];\ \ I_T^{concat} \leftarrow [I_T^{concat}, I_T^{All}[i]]$ *// concatenating the matrices*
$P_{sort}[1:nLen][1:Idx_{conCat}], indx_{sort}[1:nLen][1:Idx_{conCat}] \leftarrow sortColWise\ (\ P_T^{concat}[1:nLen][1:Idx_{conCat}]\ )$ *// sort the concatenated matrix* P_T^{concat} *column wise*
$I_{sort}[1:nLen][1:Idx_{conCat}] \leftarrow I_T^{concat}(\ indx_{sort}[1:nLen][1:Idx_{conCat}]\ )$ *// using the sorted index i.e.* $indxsort$ *rearrange* I_T^{concat}
return $P_T^{Final} \leftarrow P_{sort}[1:kNN][1:Idx_{conCat}]$ and $I_T^{Final} \leftarrow I_{sort}[1:kNN][1:Idx_{conCat}]$

Function kNN_Join_Parallel(T, μ_T, σ_T, $\mathfrak{st}$, $\mathfrak{ed}$, m)**:**

$subSeq_1 \leftarrow T[\mathfrak{st}$ to $(\mathfrak{st}+m-1)]$ {*get the* 1^{st} *subsequence*}

$[QT, D^{ignore}] \leftarrow MASS(subSeq_1, \mu_T[\mathfrak{st}], \sigma_T[\mathfrak{st}], T, \mu_T, \sigma_T)$

.....

for $i \leftarrow \mathfrak{st}$ ***to*** $\mathfrak{ed} - m + 1$ **do**

$cutSubSeq \leftarrow T[i$ to $(i+m-1)]$ {*get target subsequence by chopping* T *from index* i *to* $(i+m-1)$}

.....

return P_T and I_T {*return the* P_T *and* I_T *array* }

6.2 Utility Experiments

We illustrate the utility of kNN MP for knowledge discovery in three case studies including chemometric, accelerometer, and UCR datasets.

6.2.1 Case study: chemometric data

The experiments were performed with the dataset of 4075 spectrum, each having 680 dimensions. These spectrum represents the protein rates, measured on 10 different products: rapeseed (CLZ), corn gluten (CNG), sun flower seed (SFG), grass silage (EHH), full fat soya (FFS), wheat (FRG), sun flower seed (SFG), animal feed (ANF), soyameal (TTS), maïs (PEE), milk powder and whey (MPW). The complete data can be imagined as a matrix of 4075×680, where each row represents a time series of 680 elements. To build the kNN MP on a big time series, we concatenated the 4075 individual time series together and applied our kNN MP algorithm on it.

We created the kNN MP by considering the subsequence length $m = 40$. Fig. 4 shows the $1NN$ vs $2NN$ distances of the matches for each subsequence. Notice that we have only plotted a small part (*i.e.*, 640 elements) of the MP. A low threshold is defined to obtain the motifs (shown as dotted black line at the bottom of the curve). In the left image of Fig. 4, if we consider only the curve for $1NN$ (blue color) then there are several subsequences which can be taken as motifs. We consider these motifs as *weak motifs*. We can detect the strong motifs by checking their existence in *kNN* MP. If a subsequence appears as motif in kNN MP, that means it has at least k similar subsequences in the time series.

In Fig. 4, we see some *strong motifs* (marked as green circles) by considering $k = 2$ (left image) and $k = 4$ (right image). The motifs that appear in $4NN$ MP are stronger than those detected by using $2NN$ MP (see left image in Fig. 4) because they are repeated in higher values of k. There are some *weak motifs* (encircled by red color) that are shown in Fig 4 which appear only in *1NN* MP but not in either $2NN$ or in $4NN$ MPs. Another nice illustration is also shown in Fig. 5 where we can see the cases of *strong motifs* and *weak motifs* by considering the $1NN$ vs $2NN$ MP and $4NN$ MP.

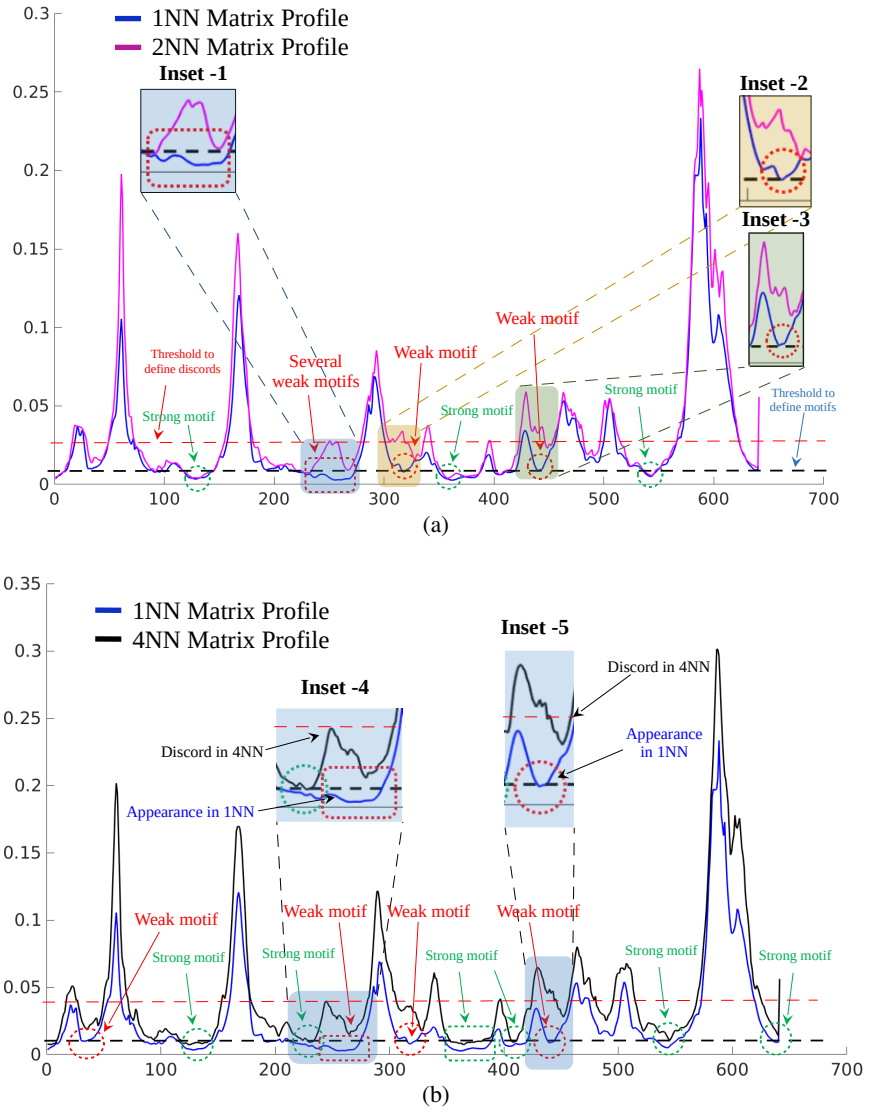
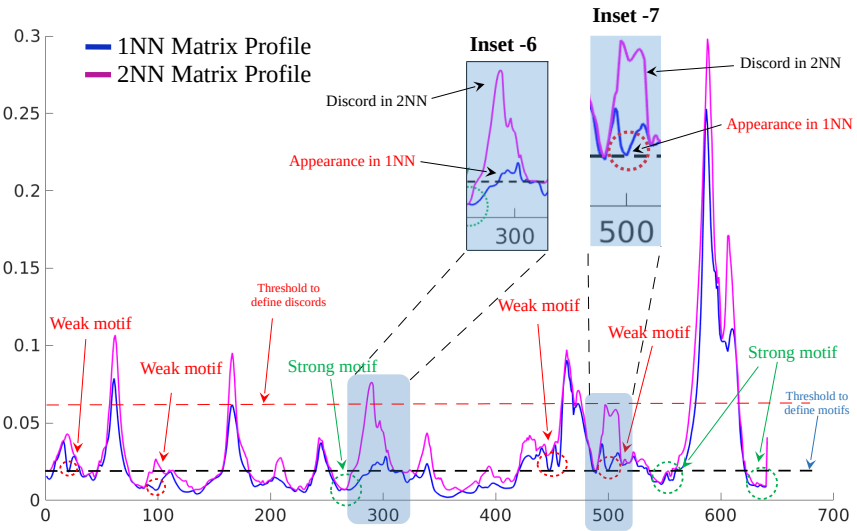
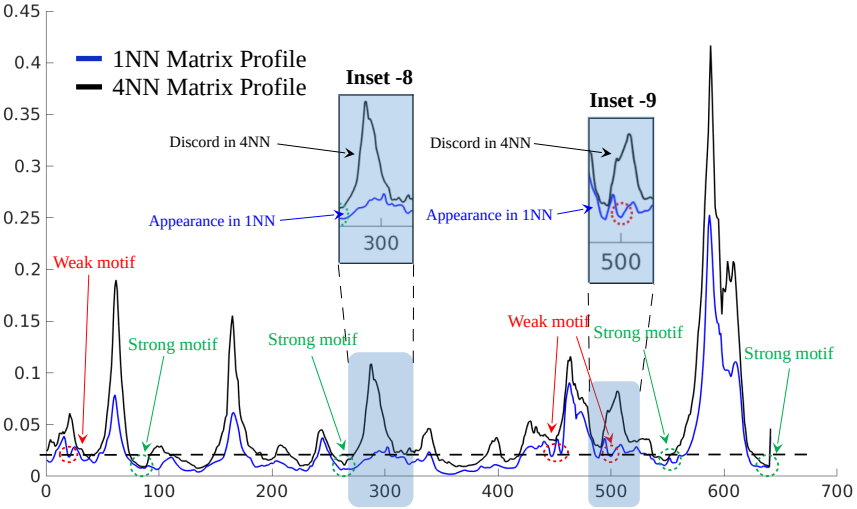


Figure 4: The extracted motifs and discords are illustrated in the plot : (a) The $1NN$ vs $2NN$ MP and (b) $1NN$ vs $4NN$ MP of a part of the time series is plotted.

The discords are the subsequences whose distance with other subsequences is high. Let's consider Fig. 6 that shows the sorted distances of the matches of three individual subsequences. The distances are shown along $Y-axis$ and the subsequences with whom the distances are computed are shown along $X-axis$. The distance values plotted in the first curve (shown in green) have high values. The points in the third curve (shown by pink color in Fig. 6) represent the sorted distance values of a



(a)



(b)

Figure 5: The extracted motifs and discords are illustrated in the plot : (a) The $1NN$ vs $2NN$ MP and (b) $1NN$ vs $4NN$ MP of a part of the time series is plotted.

subsequence. They are mostly below the defined outlier (discord) threshold. So, if we consider even $4NN$ (or even more) MP then in no way this particular subsequence will be detected as outlier (this is normal as this subsequence has close matches). But in the case of third curve, the top two distance values are less than the defined discord threshold, and the other distance values are more than the threshold. So, from

the nature of the curve it can be depicted that this particular subsequence has two close matches, but it is highly different than all other remaining subsequences. So, if we consider $1NN$ and $2NN$ then this particular subsequence will not be detected as outlier. But if we consider $3NN$, $4NN$ and more then it will be detected as an outlier. Logically, this subsequence should be detected as outlier as it has only two very close neighbors (which can be outliers), and all of it's other neighbors are very different.

This scenario is confirmed by kNN MP in Fig. 7 (top). The 1^{st} subsequence has many matches shown as pink color (Fig. 7 top), this is why $1NN$, $2NN$, $3NN$ MPs (Fig. 7 bottom) show lower value at the index of this subsequence. On the other hand, the 3^{rd} subsequence (shown in green color) has no matches, hence the $1NN$, $2NN$, $3NN$ MPs show high values for the 3^{rd} subsequence. But, for the case of 2^{nd} subsequence, it has a close match (shown in red color). Hence these two subsequences (which are marked in red color) would closely match with each other. Accordingly the $1NN$ MP (shown in Fig. 7 bottom) shows a very low value for these subsequences (follow the pink colored curve).

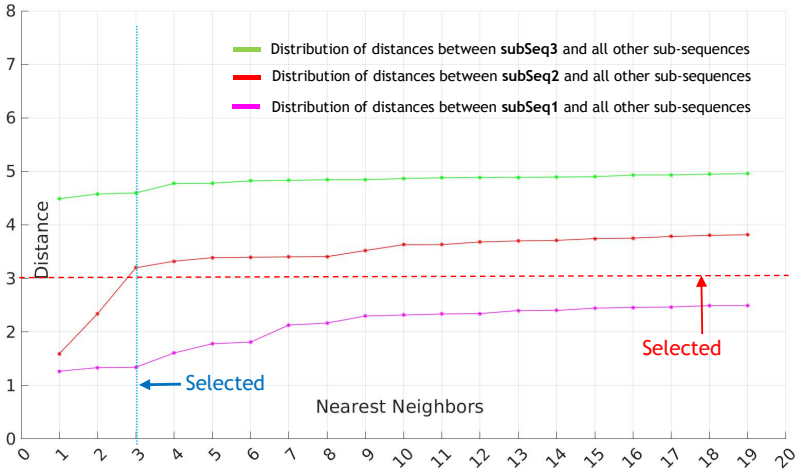


Figure 6: One special case of outliers detection

It can be seen from Fig. 7 (top) that these two red colored subsequences should be considered as outliers as they are far from all other subsequences. In fact, they are $3NN$ discords, and their detection is possible only if we compute kNN MP, for $k \geq 3$. If we calculate $2NN$ MP then each of these red colored subsequences would have the blue colored subsequence as their $2nd$ nearest neighbors (blue colored subsequence is slightly different than the red ones). Hence to detect them as outliers, we need to calculate $3NN$ MP (shown by blue color in Fig. 7 (bottom)).

Now let's again consider the example of $1NN$ vs $2NN$ and $1NN$ vs $4NN$ MPs of *chemometric dataset*. To find the discords, a relatively high threshold is taken (shown as dotted red line in Fig. 4). If we consider this threshold then the discords

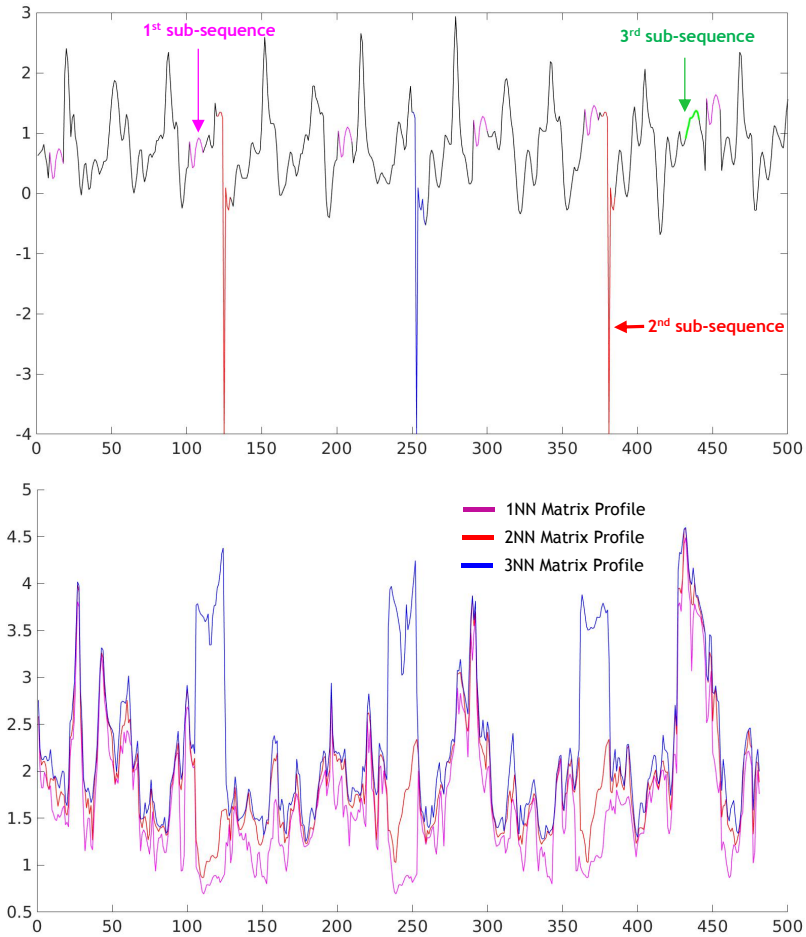


Figure 7: Special case of the outliers presence is depicted by considering a toy time series (top). The 1NN, 2NN and 3NN MPs are shown for the time series (bottom)

which are detected in $1NN$, should also be repeated (exactly at the same index) in $2NN$ as well as in $4NN$ MP. In Fig. 4 (left) there are several instances shown in Inset-1, Inset-2 and Inset-3, where we have zoomed over some portions of $1NN$ and $2NN$ MPs. It can be seen in these instances that the distance value is low (some times downward) in $1NN$ MP, but at these locations, the distance value is high in $2NN$ MP. Similar characteristics are also observed in Inset-4, Inset-5 for the case of $1NN$ vs $4NN$ MPs in Fig. 4 (right).

6.2.2 kNN similarity search : case study on accelerometer data (measured on 13 sheep)

This real world dataset corresponds to more than five thousands time series which have been measured by attaching accelerometer at the neck of 13 sheep. Acelerometers captured 3-axial acceleration at a constant rate of 100Hz. Each of the three axial acceleration gives a different information for the zoologist, but for the simplicity and to show the interest of proposed method, here we only consider X axis data. The accelerometer data are manually labeled into one of six activities: STANDING-GRAZING, STANDING-EATING BRUSH, STANDING-RUMINATING, WALKING, RUNNING, STANDING IMMOBILE. The sensor signals were pre-processed and for each activity of interest, a time series of 5 seconds (500 elements per time series) were constituted. By this manner, a dataset with 8532 time series is obtained where each of these time series is manually labeled.

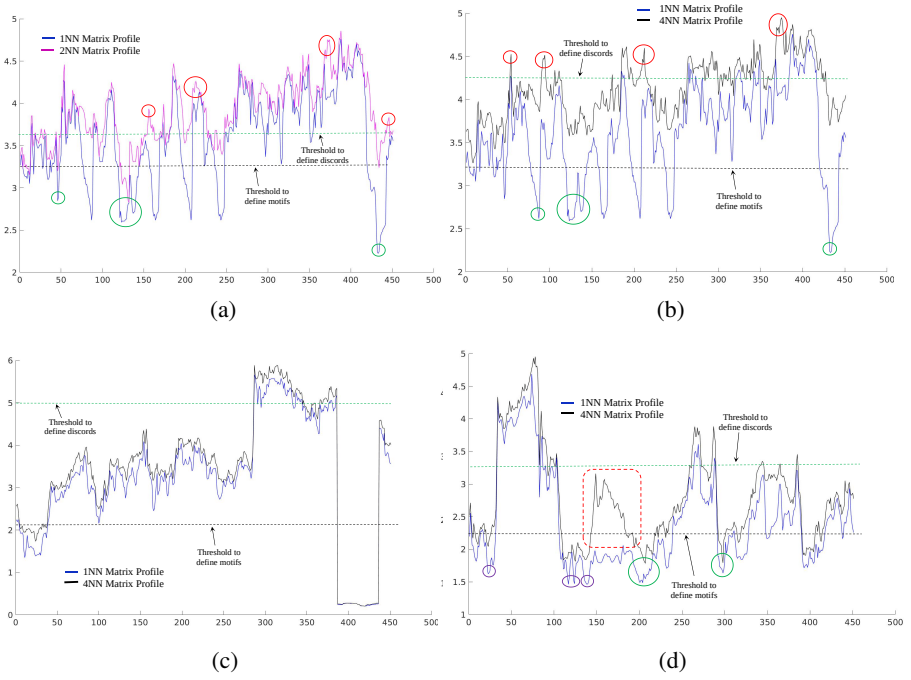


Figure 8: (a) The 1NN and 2NN MPs are plotted for part of the time series, having label as: WALKING. (b) The 1NN and 4NN MPs are plotted of the same part of the time series. (c) The 1NN and 4NN MPs (which follow very similar trajectory) are plotted for another part of the time series, having label as WALKING. (d) The 1NN and 4NN MPs are plotted for a randomly chosen part of the time series, having label as: RUNNING.

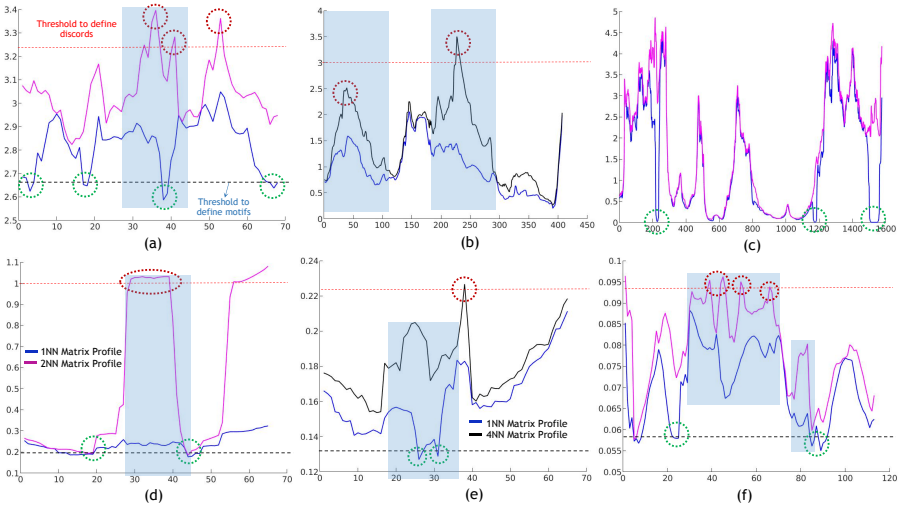


Figure 9: The results of UCR dataset : **(a)** FacesUCR dataset: the motifs and discords are shown by green and red circles respectively; **(b)** Beef dataset: presence of motifs in $4NN$ MP are not supported by $1NN$ MP; **(c)** CinCECGTorso dataset: the highlighted zone shows the contrary nature of $1NN$ vs $2NN$ MPs; **(d)** BME dataset: the highlighted zone shows the contrary nature of $1NN$ vs $2NN$ MPs; **(e)** BME dataset : some (weak) motifs in $1NN$ MP are not present in $4NN$ MPs; (f) Adiac dataset: the highlighted zones show the contrast between $1NN$ vs $2NN$ MPs where the weak motifs detected in $1NN$ MP are not present in $2NN$ MP and the discords detected in $2NN$ MP are not present in $1NN$ MP.

Here we create a big time series by concatenating all the 8532 time series and the subsequence length (m) is taken as 50. The total number of subsequences obtained from the concatenated long time series is $(500 \times 8531) - 50 + 1 = 42,65,451$. An interesting part of the kNN MP for the case of $1NN$ vs $2NN$ MP and $1NN$ vs $4NN$ MP is shown in Fig. 8a and Fig. 8b. It can be seen from these figures that at several locations the distance values in $1NN$ MP are less than the defined threshold for motif, but not in the $2NN$ MP (see the green circles). So these low points can not be considered as strong motifs. For the case of detecting outliers, several points are marked in Fig. 8a and Fig. 8b, where some outliers can be detected only in $2NN$ vs $4NN$ (see the red circled points in Fig. 8a and Fig. 8b).

In many cases, the $1NN$ and $4NN$ MPs follow almost the same trajectory and such an example is shown in Fig. 8c. Another portion of the complete MP is shown in Fig. 8d in which we illustrate the $1NN$ and $4NN$ MPs, and several interesting cases (by green and violet encircles). The points marked by green circles are *strong motifs* where both of the $1NN$ and $4NN$ MPs satisfy the defined threshold, whereas the points marked by violet circles are *weak motifs*. The region marked by dotted red rectangle shows an interesting situation where the $4NN$ MP shows a completely different trajectory (upward) than $1NN$ MP (downward). This can mainly happen

when a particular subsequence has the 1^{st} nearest neighbor with whom it has small distance, but its distance with other neighbors is high (see Fig. 6). Such subsequences are usually anomalies.

6.2.3 Case study: UCR repository

Here we show some interesting results for kNN MP by using datasets from the UCR time series archive [16] (see the list of datasets in Table 1). To create a single big time series from each dataset, we have sequentially concatenated the individual time series from training and testing set. Fig. 9 shows a portion of the generated MP for different datasets. As seen in Fig. 9 (a), (b), (d), (e), (f) at some places the $1NN$ MP shows different trajectory than kNN (*i.e.*, $2NN$ and $4NN$) MPs where the detected weak motifs (shown in dotted green circle) in $1NN$ MP are absent in kNN MP and discords (shown in dotted red circle) detected in kNN MPs are absent in $1NN$ MP (contrary to the ideal case where the detected motifs and discords will be repeated i.e. follow the same trajectory in $1NN$ as well as in kNN MPs). From these plots, we can visualize that the curve for $1NN$ MP and $2NN$ or $4NN$ MPs don't follow the same trajectory. Hence, the detection of motifs and discords from $1NN$ MP could be wrongly validated if we don't verify their confirmations (i.e. existence) in kNN (i.e. $1NN$, $2NN$...kNN) MPs. From Fig. 9 (c) also, we can observe that the detected motifs are present in only one MP (either $2NN$ or $1NN$) and are not supported by the other MPs. Probably these motifs are resulted due to the presence of noise and should be considered with precaution. Hence, from these experiments with UCR datasets, we can clearly visualize the usefulness of kNN MP over $1NN$ MP for the detection followed by confirmation of motifs and discords.

Furthermore, we report in Supplementary Materials SM: IV, our experiments of kNN MP on a large set of 87 UCR datasets.

Table 1: Dataset details from UCR archive. See Supplementary Materials SM: IV for our results on 87 UCR datasets

Dataset Name	Size of training set	Size of testing set	Time series length
FacesUCR	200	2050	131
Beef	30	30	471
CinCECGTorso	40	1380	1639
BME	30	150	128
Adiac	390	391	176

Another utility experiment for "shapelets discovery" by using kNN MP is shown in Supplementary Materials SM: III. The shapelets can be defined as the sub-sequences which can maximally represent the class of a time series. The use of kNN matrix profile help us to increase the confidence of selected sub-sequence as shapelet.

6.3 Scalability of kNN similarity search

In this section, we study the scalability of our solution for building the kNN MP, by varying several parameters such as k, number of cores, length of time series (n) and subsequence length (m). In our experiments, we used several datasets. The first

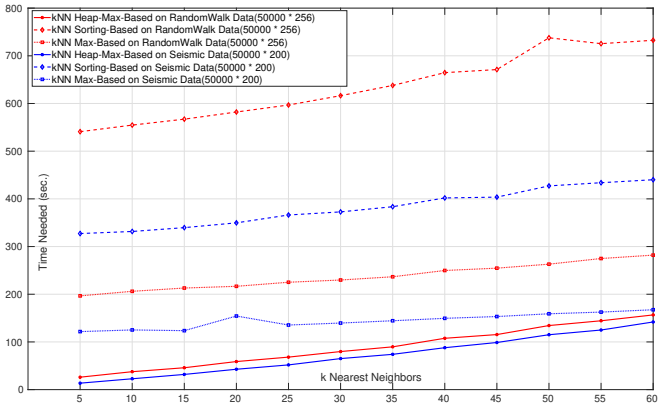


Figure 10: The variation of computational time with the increasing number of kNN similarity search. The computational time analysis is shown for three different proposed approaches, *i.e.*, max based (refer to Section 4.2.2), sort based (refer to Section 4.2.1) and heap-max based (refer to Section 4.2.3) for kNN similarity search.

one is called as *seismic* dataset, obtained from the domain of seismology, containing 50000 individual time series. The length of each time series is 200. By concatenating all time series, we obtained a big time series of 1 million values. The second dataset is called *random walk* dataset, having 50000 individual time series. The length of each time series is 256. We also concatenated the time series of this database to create a big time series. The initial three experiments (Fig. 11a, Fig. 11b, Fig.11c) were performed by using these two datasets. For the experiments on the number of cores in (Fig. 11d), we have used the *Hyper-spectral data of protein levels* and *accelerometer* datasets (which are are used in other experiments, mentioned in Section 6.2.1 and 6.2.2). The primary reason of choosing these datasets is the higher length of their time series (680 and 500 respectively). As like the previous datasets, here also we have concatenated all the time series to build a big time series. In our experiments the default value for k is taken as 10. For the first experiment, we incremented the value of k to study its effect on the time required for computing kNN MP. As seen in Fig. 11a, with increasing k, the required computational time increases linearly for both datasets. But, the computational time doesn't increase drastically for higher values of kNN and the proposed algorithm is able to give output of high kNN values with small change in computational time. From the second experiment shown in Fig. 11b, it can be seen that the computational time increases with increasing time

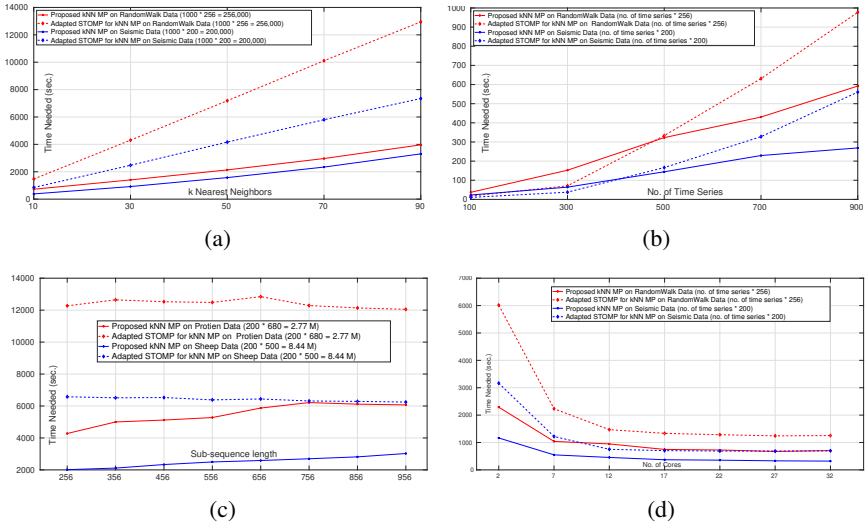


Figure 11: The comparative performance of our proposed algorithm and the technique by Yeh et.al [2] on random-walk, seismic, protein and sheep datasets: (a) The variation of computational time with the increasing k for generating kNN MP. The color red and blue represents the plot for random-walk and seismic datasets respectively. (b) The variation of computational time with the increasing length of time series (x-axis is representing number of individual time series are concatenated to generate a single big time series). (c) The variation of computational time with the increasing length of subsequence (m) for the protein and sheep datasets respectively. (d) The computational time with the increasing the number of cores for the random-walk and seismic datasets respectively.

series length. In Fig. 11c, it can be seen that computational time is almost linear with increasing the subsequence length.

Fig. 11d shows the evolution of the computational time with increasing the number of cores. As seen, the computational time firstly decreases significantly with the increase of number of cores for both datasets, but after a certain number of cores, it gets constant (after 17 cores).

6.3.1 Comparison with adapted STOMP

We have compared the computational time of STOMP algorithm by Yeh et.al [2] with our proposed technique for kNN MP. In general, the STOMP algorithm is built for computing $1NN$ MP. We adapted it for kNN MP by applying it in k iterations, such that the best matches obtained in each iteration are excluded before the next iteration. Fig. 11 shows the computational time of our proposed algorithm and STOMP algorithm by using the previously mentioned *seismic* and *random walk* datasets. The variation of computational time is depicted with the increase of k by using 1000 individual time series from each datasets (seismic and random walk datasets, shown

in Fig. 11a). It can be seen that our algorithm outperforms significantly the STOMP technique for computing kNN MP.

The second experiment, shown in Fig. 11b is performed to compare the computational time with the increase of time series length. The *x-axis* in the plot represents the number of individual time series which are concatenated to generate the single big time series to calculate *kNN MP*. In this experiment, our proposed technique has outperformed the STOMP algorithm. In the third experiment (shown in Fig. 11c), where the comparison is performed by increasing the subsequence length, our proposed technique has also better execution time than the STOMP algorithm, for computing kNN matrix profile.

Further comparisons with other MP techniques, e.g., *SCRIMP* & *SCRIMP++* are mentioned in Supplementary Materials SM: II. Due to the complexity of adapting *SCRIMP* & *SCRIMP++* techniques for parallel computation of kNN MP on multiple cores, we have used adapted these techniques for sequential computation on a single core on comparatively smaller dataset. Furthermore, we have also executed the adapted STAMP/STOMP technique and our proposed algorithm on a single core for the purpose of fair comparison.

7 Conclusion

In this paper, we proposed an efficient technique for computing *kNN MP*. Our technique is fast, simple and parallelizable on multiple cores of an off-the-shelf computer. Using several real datasets, we showed that our proposed *kNN MP* can be more useful than *1NN MP* for motif and outlier discovery from time series datasets. Through our experimental evaluation, we illustrated the effectiveness of generating *kNN MP* by our technique. As a future work, we plan to extend our parallel technique for GPUs. Moreover, we would like to extend the *kNN MP* for the case of multi-dimensional data.

Acknowledgment

We greatly acknowledge the funding from *Safran Data Analytics Lab*. The authors are grateful to Inria Sophia Antipolis - Méditerranée "Nef" computation cluster for providing resources and support.

Appendix: I Fast Calculation of Distance

Mueen et. al proposed a technique, known as *Mueen's Algorithm for Similarity Search (MASS)* [9] [8] for the fast calculation of *Euclidean Distance* between query subsequence and the subsequent subsequence of target time series, by exploiting the *Fast Fourier Transform (FFT)* (time complexity is $O(n\ logn)$). The technique is mentioned in following Algorithm 6.

In Line 4, both vectors q and T are made to be of the same length (see Section Appendix: I.1) by appending the required amount of zeros $(2n - m)$ to the reversed query so that like T_a, q_{ra} will have $2n$ elements (to facilitate element wise

Algorithm 6: SLIDINGDOTPRODUCT

Input: A query subsequence (q), A target time series (T)
Output: The dot product (qT) between single query subsequence and all the target subsequences

1 $n \leftarrow$ no. of elements in T; $s \leftarrow$ no. of elements in q
2 $T_a \leftarrow$ double the length of T by appending n number of zeros at the end
3 $q_a \leftarrow$ reverse the elements in q so that the last element become first and vice versa
4 $q_{ra} \leftarrow$ append $(2n - m)$ zeros at the end of q_r
5 $q_{raf} \leftarrow$ do $FFT(q_{ra})$; $T_{af} \leftarrow$ do $FFT(T_a)$
6 $M \leftarrow$ elementwise multiplication of q_{raf} and T_{af} *// as both q_{raf} and T_{af} are of the same size, so we can easily multiply element to elements*
7 $qT \leftarrow InverseFFT(M)$
8 **return** P_T

multiplication in frequency domain). In Line 5, we perform the Fourier transform (time complexity is $O(n \ \log n)$) of q_{ra} and T_a to transform time domain signals into frequency domain signals. Then in Line 6, an element wise multiplication is performed between two complex valued vectors, followed by inverse FFT on the resultant product.

Appendix: I.1 Relation between convolution in time domain with frequency domain

The time domain convolution of two signals $T = [T_1, T_2, ..., T_n]$ and $q = [q_1, q_2, ..., q_m]; m << n$ can be calculated by sliding q upon T (for implementation, we would need $\frac{m}{2}$ number of zeros padding at the beginning and at the end of original T vector). The convolution between these two vectors are represented by $(T * q)$ which is a vector $\mathbf{c} = [c_1, c_2, ..., c_{n+m-1}]$ denoted as : $\mathbf{C}_p = \sum_u T_u \ q_{p-u+1}$ where $u = [\max(1, p - u + 1)].....[\min(n, m)]$ and u ranges over all the sub-scripts for T_u and q_{p-u+1}.

The convolution in time domain can be quickly calculated by element-wise multiplication in frequency domain by taking *Fourier Transform* of two signals, multiply them element-wise and then applying inverse Fourier transform. Performing full convolution (in both time and frequency domains) between two $1D$ (same for $2D$ also) signals of size n and m results in an output of size $(n + m - 1)$ elements. Therefore, two signals are made of same size by padding zeros at the end to facilitate the multiplication operation.

Appendix: II Fast Calculation of Mean and Standard Deviation :

The fast calculation of mean (μ) and standard deviation (σ) of a vector of elements (x) is proposed by Rakthanmanon et.al [17]. The technique needs only one scan

through the sample to compute the mean and standard deviation of all the subsequences. The mean of the subsequences can be calculated by keeping two running sums of the long time series which have a lag of exactly m values.

$$\mu = \frac{1}{m}\left(\sum_{i=1}^{k} x_i - \sum_{i=1}^{k-m} x_i\right) \quad \sigma^2 = \frac{1}{m}\left(\sum_{i=1}^{k} x_i^2 - \sum_{i=1}^{k-m} x_i^2\right) - \mu^2 \quad (1)$$

In the same manner, the sum of squares of the subsequences can also be calculated which are used to compute the standard deviation of all the subsequences by using the Equations 1.

Appendix: III Brief description of MASS algorithm :

The *MASS* algorithm is mentioned in Algorithm 7. In Line 1 of this algorithm, we calculate the sliding dot product by using the Algorithm 6.

Algorithm 7: MASS $(q, \mu_q, \sigma_q, T, \mu_T, \sigma_T)$

Input: A query subsequence (q), mean of q (μ_q), standard deviation of q (σ_q), Target time series (T), mean of T (μ_T), standard deviation of T (σ_T)

Output: Distance profile (D), Dot product (qT)

1 $qT \leftarrow SlidingDotProducts(q, T)$ *//see Algorithm 6*

2 $D \leftarrow CalculateDistanceProfile(qT, \mu_q, \sigma_q, \mu_T, \sigma_T)$ *// see Equation 2*

3 **return** qT, D

After that the z-normalized Euclidean distance $(D[i])$ is calculated between two time series subsequences q and $T_{i,m}$ by using the dot product between them $(qT[i])$. The formula to calculate the distance $(D[i])$ is shown below.

$$D[i] = \sqrt{2m\left(1 - \frac{qT[i] - m\mu_q\mu_T[i]}{m\sigma_q\sigma_T[i]}\right)} \quad (2)$$

In this Equation 2, m is the subsequence length, μ_q is the mean of query sequence q, $\mu_T[i]$ is the mean of $T_{i,m}$, σ_q is the standard deviation of q and $\sigma_T[i]$ is the standard deviation of $T_{i,m}$.

Appendix: IV Brief description of STAMP algorithm :

The *Scalable Time Series Anytime MP (STAMP)* algorithm, proposed by Yeh et.al [2] (outlined in Algorithm 8) calculates the closest match (*1NN*) of every subsequence in a time series T, based on the calculated distance (called as *distance profile*) between any particular subsequence with all the remaining subsequence in T. The algorithm is mentioned in Algorithm 8. A for loop is run in Line 6 to chop each subsequence (consider as *query* for better understanding). Then a distance vector $(Dist_{cutQuery})$

Algorithm 8: STAMP(T, m)

Input: The user given time series T, subsequence length m
Output: The MP P_T and associated matrix profile index I_T

1 $n_T \leftarrow length(T)$ *// get the no. of elements in T*
2 $P_T \leftarrow$ Initialize this $1D$ vector with inf
3 $I_T \leftarrow$ Initialize this $1D$ vector with zeros
4 $Idxs \leftarrow (n_T - m + 1)$ *// total number of possible subsequences*
5 $\mu_T, \sigma_T \leftarrow ComputeMeanStd(T)$ *// see Equation 1*
6 **for** $i \leftarrow 1$ ***to*** $Idxs$ **do**
7 $cutQuery \leftarrow T[i$ to $(i + m - 1)]$ *// get query subsequence by chopping T from index i to $(i + m - 1)$*
8 $Dist_{cutQuery} \leftarrow MASS(cutQuery, \mu_T[i], \sigma_T[i], T, \mu_T, \sigma_T)$ *// calculate the distance between query subsequence and the other subsequences in T*
9 $P_T \leftarrow computeElementwiseMin(Dist_{cutQuery}, P_T)$ *// perform element-wise minimum of values from P_T and $Dist_{cutQuery}$ and store the minimum value in P_T*
10 $I_T \leftarrow i$ *//update I_T at the indexes where element-wise minimum operation replaces the previously stored value in P_T with the new value from $Dist_{cutQuery}$*
11 **return** P_T and I_T *// return the P_T and I_T array*

is computed by calculating the distances between all other subsequences in T and the *query*. In each iteration, the smallest distance and their corresponding indexes are iteratively kept in P_T and I_T vectors by performing the element-wise or index-wise comparison between these two vectors P_T and $Dist_{cutQuery}$. To introduce the anytime nature of *STAMP* algorithm, we just need to randomly choose the index of query subsequence from T (in Line 6 of Algorithm 8).

Appendix: V Brief description of STOMP algorithm :

The *Scalable Time Series Ordered Search MP (STOMP)* algorithm [11] is a variant of *STAMP* in which we perform an ordered search (from left to right) and unlike anytime version of *STAMP*, we can't perform unordered or random search. To calculate distance, *STOMP* algorithm takes benefit of the common part between two adjacent subsequences which is same except the fist and last elements. The Z-normalized euclidean distance ($D_{i,j}$) between two time series subsequences $T^q_{j,m}$ and $T_{i,m}$ is calculated by using the Equation 2. The dot product between these two subsequences are mentioned as $QT_{i,j}$ and based on these notations, the Equation 2 can be rewritten as :

$$D_{i,j} = \sqrt{2m\left(1 - \frac{QT_{i,j} - m\mu_i\mu_j}{m\sigma_i\sigma_j}\right)} \tag{3}$$

where m is the subsequence length, μ_j is the mean of $T^q_{j,m}$, μ_i is the mean of $T_{i,m}$, σ_j is the standard deviation of $T^Q_{j,m}$ and σ_i is the standard deviation of $T_{i,m}$. Following below, we explain how $QT_{i,j}$ can be computed in $\mathcal{O}(1)$ time when

$QT_{i-1,j-1}$ has already been calculated. The term $QT_{i-1,j-1}$ can be decomposed as : $QT_{i-1,j-1} = \sum_{k=0}^{m-1} T_{i-1+k}T^Q_{j-1+k}$ and the term $QT_{i,j}$ can be decomposed as : $QT_{i,j} = \sum_{k=0}^{m-1} T_{i+k}T^Q_{j+k}$ Thus by combining these two terms we can get : $QT_{i,j} = QT_{i-1,j-1} - T_{i-1}T^Q_{j-1} + T_{i+m-1}T^Q_{j+m-1}$. The relationship between $QT_{i,j}$ and $QT_{i-1,j-1}$ indicates that from the distance profile of query subsequence $T^Q_{j-1,m}$, we can compute the distance profile of $T^Q_{j,m}$ in $\mathcal{O}(1)$ time.

Appendix: V.1 Two independent time series matching using STOMP

In the above section, we have explained the general principal of *STOMP* algorithm for the computation of MP for a single time series. In case of two independent time series, the basic *STOMP* algorithm needs to be marginally modified. The pseudo code of modified *STOMP* algorithm, named as *IndependentSTOMP* is mentioned in Algorithm 9.

While called, this algorithm calculates the distances from 2 query subsequence until the last query subsequences *i.e.*, $(Idxs)$ (see Line 2) because before calling this algorithm, we have already computed the distances between 1^{st} query subsequence and target subsequence t. Now let's concentrate on the basic principal of

Algorithm 9: INDEPENDENTSTOMP(t, μ_t, σ_t, $tqSingleVal$, Q, QT, μ_Q, σ_Q)

Input: One target subsequence (t), Mean of t (μ_t), Standard deviation of t (σ_t), Dot product value between first query subsequence and t $(tqSingleVal)$, Query time series (Q), Existing dot product between all query subsequences and the previous target subsequence (QT), Mean of whole query time series Q (μ_Q), Standard deviation of Q (σ_Q)

Output: A MP P_Q and associated MP index I_Q

1 $Idxs \leftarrow (n_Q - m + 1)$ *// total number of possible subsequences*

2 **for** $j \leftarrow 2$ ***to*** $Idxs$ **do**

3 $\quad QT[j] \leftarrow QT[j-1] - (Q[j-1] \times t[1]) + (Q[j+m-1] \times t[1+m-1])$

4 $QT[1] \leftarrow tqSingleVal$

5 $Dist_{cutQuery} \leftarrow CalculateDistanceProfile(QT, \mu_t, \sigma_t, \mu_Q, \sigma_Q)$ *// calculate the distance profile*

6 **return** $Dist_{cutQuery}$ and QT *// return the $Dist_{cutQuery}$ and QT array*

STOMP algorithm. Remind that the dot product profile (QT) of any particular subsequence, can be derived from the dot product profile of it's previous subsequence (see Section Appendix: V). So, in Line 3 we repetitively calculate the dot product profile between each query subsequence (from 2^{nd} subsequence onward) and the target subsequence (t). The 1^{st} term *i.e.* $(QT[j-1])$ at the right hand side of Line 3

holds the dot product profile of the previous target subsequence (in comparison with t) and j^{th} query subsequence. The $Q[j-1]$ in 2^{nd} term represents the 1^{st} element of the previous query subsequence and $t[1]$ represents the 1^{st} element of the target subsequence (t). In the same manner, $Q[j+m-1]$ and $t[1+m-1]$ terms represents the last element of j^{th} query subsequence (current one) and the last element of the subsequences t. In Line 4, the dot product value between first query subsequence $(j=1)$ and the specific target subsequence (t) is copied in $QT[1]$. This value is taken as input in this algorithm (*i.e.*, $tqSingleVal$) The distance profile is calculated in Line 5 by using the dot product profile (QT), mean (μ_t) and STD (σ_t) value of target subsequence t, mean (μ_Q) and STD (σ_Q) vector of query time series. Finally the distance vector $(Dist_{cutQuery})$ and the dot product vector (QT) are returned as the output from the algorithm.

The time complexity of classical *STOMP* [11] is $\mathcal{O}(n)^2$ which is $\mathcal{O}(\log n)$ improvement over *STAMP* algorithm which is an optimal improvement for a fully join algorithm and compared to small dataset, this improvement is highly useful for large datasets.

References

[1] Yeh, C.-C.M., Herle, H.V., Keogh, E.J.: Matrix Profile {III:} The Matrix Profile Allows Visualization of Salient Subsequences in Massive Time Series. In: Proceedings of the International Conference on Data Mining (ICDM), pp. 579–588 (2016)

[2] Yeh, C.C.M., Zhu, Y., Ulanova, L., Begum, N., Ding, Y., Dau, H.A., Zimmerman, Z., Silva, D.F., Mueen, A., Keogh, E.: Time Series Joins, Motifs, Discords and Shapelets: a Unifying View that Exploits the Matrix Profile vol. 32, pp. 83–123. Springer, ??? (2018)

[3] 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. IEEE, ??? (2018)

[4] Sinha, S.: Discriminative motifs. In: Proceedings of the Sixth Annual International Conference on Computational Biology, {RECOMB} 2002, Washington, DC, USA, April 18-21, 2002, pp. 291–298 (2002)

[5] Balasubramanian, A., Wang, J., Prabhakaran, B.: Discovering Multidimensional Motifs in Physiological Signals for Personalized Healthcare. J. Sel. Topics Signal Processing **10**(5), 832–841 (2016)

[6] Yagoubi, D.E., Akbarinia, R., Kolev, B., Levchenko, O., Masseglia, F., Valduriez, P., Shasha, D.E.: ParCorr: efficient parallel methods to identify similar time series pairs across sliding windows. Data Mining and Knowledge Discovery (DMKD) **32**(5), 1481–1507 (2018)

[7] Campana, B., Rakthanmanon, T., Batista, G., Mueen, A., Zhu, Q., Keogh, E., Westover, B., Zakaria, J.: Searching and mining trillions of time series subsequences under dynamic time warping, 262 (2012)

[8] Mueen, A., Keogh, E., Young, N.: Logical-shapelets: An expressive primitive for time series classification. Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 1154–1162 (2011)

[9] Mueen, A., Hamooni, H., Estrada, T.: Time Series Join on Subsequence Correlation. Proceedings - IEEE International Conference on Data Mining, ICDM (January), 450–459 (2014)

[10] Yeh, C., Zhu, Y., Ulanova, L., Begum, N.: Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View that Includes Motifs, Discords and Shapelets. Ieee (2016)

[11] 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. Proceedings - IEEE International Conference on Data Mining, ICDM, 739–748 (2017)

[12] Zhu, Y., Yeh, C.-C.C.M., Zimmerman, Z., Keogh, E.J.: Matrix Profile {XVII:} Indexing the Matrix Profile to Allow Arbitrary Range Queries. Proceedings - International Conference on Data Engineering **2020-April**, 1846–1849 (2020)

[13] Nakamura, T., Imamura, M., Mercer, R., Keogh, E.: MERLIN: Parameter-Free Discovery of Arbitrary Length Anomalies in Massive Time Series Archives (2020)

[14] Sara, M., Alireza, A., Shailendra, A., Amy, S., Eamonn, M.: Matrix Profile XXIII : Contrast Profile : A Novel Time Series Primitive that Allows Real World Classification (2021)

[15] Zhu, Y., Zimmerman, Z., Senobari, N.S., Yeh, C.-C.M., Funning, G., Mueen, A., Brisk, P., Keogh, E.J.: Matrix Profile {II:} Exploiting a Novel Algorithm and GPUs to Break the One Hundred Million Barrier for Time Series Motifs and Joins. In: Proceedings of the International Conference on Data Mining (ICDM), pp. 739–748 (2016)

[16] Dau, Hoang Anh and Keogh, Eamonn and Kamgar, Kaveh and Yeh, Chin-Chia Michael and Zhu, Yan and Gharghabi, Shaghayegh and Ratanamahatana, Chotirat Ann and Yanping and Hu, Bing and Begum, Nurjahan and Bagnall, Anthony and Mueen, Abdullah and Batista, Gustavo, and Hexagon-ML: The UCR Time Series Classification Archive. https://www.cs.ucr.edu/~eamonn/time_series_data_2018/ (2018)

[17] Rakthanmanon, T., Campana, B., Mueen, A., Batista, G., Westover, B., Zhu, Q., Zakaria, J., Keogh, E., Riverside, U.C., Hospital, W., Paulo, S.: Searching and mining trillions of time series subsequences under dynamic time warping. Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '12, 262 (2012)

[18] Rakthanmanon, T., Keogh, E.: Fast shapelets: A scalable algorithm for discovering time series shapelets. In: Proceedings of the 2013 SIAM International Conference on Data Mining, SDM 2013 (2013)

[19] Yanping, C., Keogh, E., Hu, B., Begum, N., Bagnall, A., Mueen, A., Batista, G.: UCR Time Series Classification Archive. URL:www.cs.ucr.edu/~eamonn/time_series_data/ (2015)

Supplementary Materials (SM)

SM: I Case studies of multi-core based parallel computing

As mentioned in Section 5 that we propose an approach to perform the parallel computation of kNN MP by exploiting multiple cores. In the following section, we discuss about two spacial cases:

1. Handling the matches, coming from two independent and individual time series
2. Handling the matches, coming from two border files, treated by two separate cores of a computer

SM: I.1 Case 1 : Matches coming from two different time series

The subsequence matches can entirely come from a single individual time series (remember if several individual time series were concatenated to form a big time series T) or the matched subsequence may come from two separate individual time series. Hence, if this condition $(\mathfrak{t} + m) - 1 > \mathfrak{n}$ is satisfied for any matching subsequence then it can be confirmed that this subsequence is formed by the concatenation of two individual time series; where $\mathfrak{t}$ is the index of matching subsequence within individual time series, m is the subsequence length and $\mathfrak{n}$ ($m << \mathfrak{n}$) is the length of the individual time series. The initial part from the first individual time series will start from $\mathfrak{t}$ and will end at $\mathfrak{n}$. The remaining last part will be obtained from the next time series which will definitely start from index 1 and will continue the length equal to the remaining part of the subsequence *i.e.*, $[m - (\mathfrak{n} - \mathfrak{t} + 1)]$. Such a scenario is depicted in Fig. 12a. Otherwise, if the subsequence comes from a single time series then the start and end indexes are simply obtained as $\mathfrak{t}$ and $\mathfrak{t} + m$ respectively.

SM: I.2 Case 2 : Matches obtained from two border time series which are treated by separate cores

As mentioned before the full time series T is divided into g parts and each part is handled by separate cores. If we think carefully then we can understand that the last portion of a part, say $\mathfrak{p}^{th}$ (where $\mathfrak{p} \in g - 1$) part[5] was never concatenated with the initial portion of $(\mathfrak{p} + 1)^{th}$ part. Hence the extra subsequences generated by this concatenation were never examined before. If we inspect in more details, then we can understand that after concatenating these two subsequent parts of the time series, all the previous subsequences until *Last subsequence* (see Fig. 12b) in the first part and all the posterior subsequences, starting from *the first subsequence* (see Fig. 12b) were already considered. But the newly generated subsequences after *the last subsequence* haven't yet been explored (the subsequences in blue, green, violet and orange color in Fig. 12b). So, we only chop out this portion of unexplored data and *HeapMax-Based-kNN* algorithm (see Algorithm 4) is used to find kNN matches with respect to all

[5] $g - 1$ because the objective is to merge one part of a time series with the next part

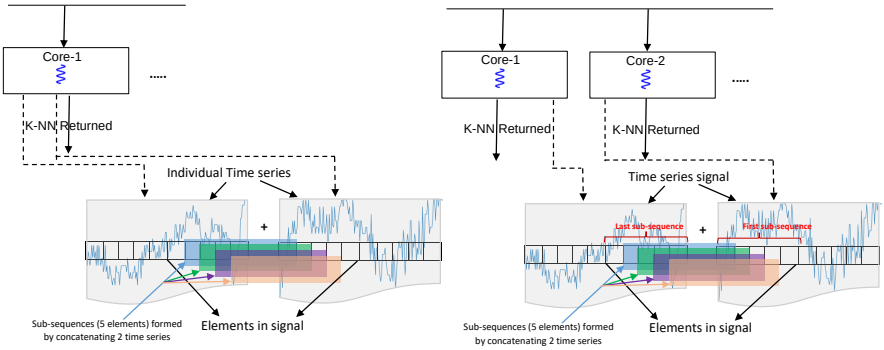


Figure 12: The analysis of computational time of the proposed algorithm on random-walk and seismic datasets: (a) The proposed technique to manage the matching result comes from two different files in multi core based parallel processing architecture. Given the size of subsequence, equal to 5, the subsequences generated by the combinations of two files are shown in color blue, green, violet and orange. (b) The proposed technique to handle the matched subsequence which belongs to the juncture, formed by concatenating the last portion of a divided time series part and the initial portion of the next part of the divided time series, where both of these divided parts of the time series are handled by separate cores. These newly generated subsequences which were not yet explored before are shown in blue, green, violet and orange color.

the subsequence of T which returns the kNN distances and their corresponding indexes. We keep accumulating such kNN distances and their corresponding indexes for all couples of time series parts in two arrays; say $P_{\mathtt{merge}^{border}}$ and $I_{\mathtt{merge}^{border}}$ respectively. Then from the $P_{\mathtt{merge}^{border}}$ array, the final top kNN distances (say $pFinalKnn_{\mathtt{merge}^{border}}$) and their corresponding indexes (from $I_{\mathtt{merge}^{border}}$) are obtained. The overall kNN matches are computed by considering these distances from $pFinalKnn_{\mathtt{merge}^{border}}$ and P_{TFinal} arrays (see Algorithm 5) together. [6]

SM: II Further comparison with adapted STOMP, SCRIMP & SCRIMP++

The kNN MP algorithm is adapted in the following manner for the case of $STOMP$, $SCRIMP$ and $SCRIMP++$ algorithms. Unlike the $STOMP$ algorithm which could be easily executed in parallel (by using multiple cores. See Algorithm 5 and Section 5 for more details) for bigger dataset, the $SCRIMP$ and $SCRIMP++$ algorithms are bit complicated to execute in parallel and needed important modifications of the actual algorithms (which is outside of the scope of this article). Hence,

[6]Please note that the above mentioned two cases are proposed by considering that a big time series T is formed by concatenating several small individual time series. If the big time series is already available (where concatenation were not needed) then the first case will not be useful.

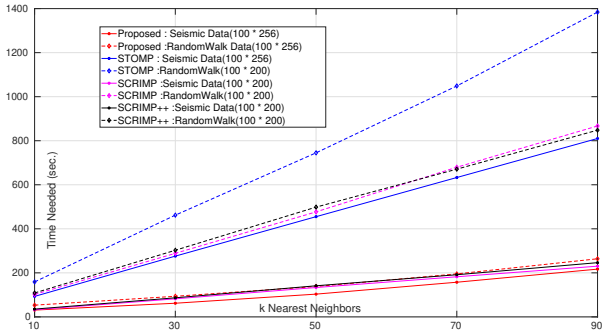
we choose to execute the $SCRIMP$ and $SCRIMP++$ algorithms in single core without any parallel processing. For the comparison purpose, we have also executed the proposed kNN MP algorithm and adapted kNN MP for STOMP algorithm using the same configuration i.e. by using single core and sequential computing. The pseudo code of this technique is mentioned in Algorithm 1.

To find kNN MP, we have iteratively called the actual $STOMP$ or $SCRIMP$ & *SCRIMP++* algorithms for kNN number of times. Generally, $STOMP$, $SCRIMP$ & *SCRIMP++* algorithms returns $1NN$ MP and index profile from the given time series (T). To avoid the consideration of same sub-sequence as the $1NN$ of any particular sub-sequence (during the computation of MP in each iteration), we have used a $1D$ array of size $Idxs$, named as $Flag$ which is initialized with zeros. Values at certain indexes of $Flag$ are modified to 1 by using the indexes stored in I_P variable, corresponding to $1NN$ MP (calculated kNN number of times).

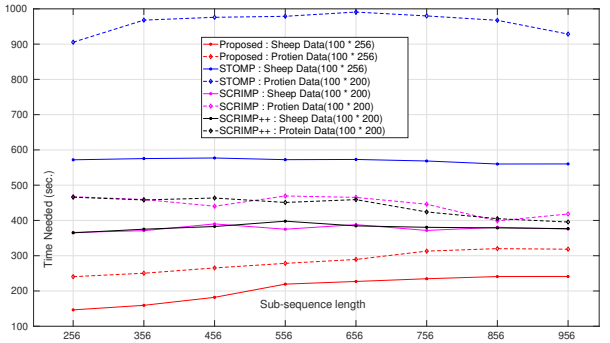
Thanks to this $Flag$ array, in each iteration, only those sub-sequences are considered in $STOMP$, $SCRIMP$ & *SCRIMP++* algorithms to compute MP, where the values at the corresponding indexes in $Flag$ variable are containing 0s. In this way, we can simply avoid the redundant consideration of the same sub-sequence for the computation of MP. It will avoid redundant consideration of same sub-sequence among any of the k nearest neighbors during the matching with any particular sub-sequence in the time series. Simply speaking, we are removing all the sub-sequences of $1NN$ MP to be considered (which are obtained in previous iteration) for the calculation of MP in next iteration.

Obviously, it isn't the correct way because during the computation of MP in any particular iteration (we have total k iterations to find kNN MP) if the t^{th} sub-sequence is appeared as the closest neighbor of p^{th} sub-sequence in the MP, hence according to our process, we are completely avoiding to consider this t^{th} sub-sequence to become any of the $p^{th}(p \in \{1....\mathcal{K}\})$ nearest neighbor of any other sub-sequence during the calculation of MP in the next iteration. Instead of that, we should have only constraint the t^{th} sub-sequence to appear again as the closest neighbor of p^{th} sub-sequence during the calculation of MP in the next iterations. But, that would have further increase the complexity of the algorithm. Moreover, our goal of calculating the computational time of the kNN MP could be easily served in this manner. But as the inconvenience of performing such technique, we are actually not finding the correct kNN MP and this technique can only serve us to compute an approximate computational time. On the contrary, our proposed algorithm provides the correct kNN MP and the computational time of our technique is compared (in the same plot) with the ones of $STOMP$, $SCRIMP$ and *SCRIMP++*. By using this manner of computing the computational time, we are favoring (because they are not providing the correct kNN MP) the $STOMP$, $SCRIMP$ and $SCRIMP++$ algorithms compared to our proposed technique.

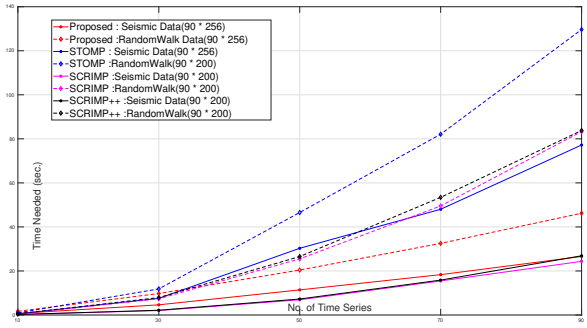
In this section, we study the scalability of our proposed algorithm in comparison to other rival algorithm such as *STOMP* [2], *SCRIMP* and *SCRIMP++* [2] for computing the kNN MP, by varying several parameters such as k, length of time series (n) and subsequence length (m). As mentioned before that due to the algorithmic constraints of *SCRIMP* and *SCRIMP++* techniques, all of the four algorithms are



(a)



(b)



(c)

Figure 13: The comparative performance of our proposed algorithm, STOMP algorithm by Yeh et. al [2] and SCRIMP & SCRIMP++ algorithms by Zhu et.al [3]. The red, blue, pink and black colors represent the proposed, $STOMP$, $SCRIMP$ and $SCRIMP++$ algorithms in the plots : (a) The variation of computational time with the increasing k for generating kNN MP. The continuous and dotted lines represent the plot for seismic and random-walk datasets respectively. (b) The variation of computational time with the increasing length of of subsequence (m) for the protein and sheep datasets respectively. The continuous and dotted lines represent the plot for sheep and protein datasets respectively. (c) The variation of computational time with the increasing length of time series (x-axis is representing number of individual time series are concatenated to generate a single big time series).

Algorithm 1 kNN_STOMP_SCRIMP(T, m)

Input: The user given time series T, subsequence length m
Output: The MP P_T and associated matrix profile index I_T

$n_T \leftarrow length(T)$ *// get the no. of elements in T*
$Idxs \leftarrow (n_T - m + 1)$ *// total number of possible subsequences*
$Idxs \leftarrow (n_T - m + 1)$ *// total number of possible subsequences*
$Flag \leftarrow 0$ *// Initialize this 1D vector with Idxs no. of zeros*
for $k \leftarrow 1$ ***to*** kNN **do**
  $[P_T, I_T] \leftarrow STOMP(T, m, Flag)$ *// $[P_T, I_T] \leftarrow SCRIMP(T, m, Flag)$*
  $[store_k^{P_T}, store_k^{I_T}] \leftarrow P_T, I_T$ **for** $i \leftarrow 1$ ***to*** $Idxs$ **do**
    $p \leftarrow I_T[i]$ $Flag_p \leftarrow 1$
return $store_k^{P_T}$ and $store_k^{I_T}$

executed in sequential manner on a single core of the computer to have a fair comparison between them. Similar to the experimental protocols used in Section 6.3, here also we have used same datasets for our experiments. We have used *seismic* and *random walk* datasets for the experiments of computing the kNN MP by varying the parameter k (see Fig. 13a) and the length of time series (n) (see Fig. 13c). Whereas the *accelerometer* and *Hyper-spectral data of protein levels* datasets are used for the experiments of computing the kNN MP by varying the subsequence length (m) (see Fig. 13b). The primary reason of choosing these datasets is the higher length of their time series (500 and 680 respectively). Same as the previous datasets, all the time series are concatenated to build a big time series and the default value for k is taken as 10.

For the first experiment, we incremented the values of k to study its effect on the time required for computing kNN MP. As seen in Fig. 13a that for all the algorithms, with the increasing k, the required computational time increases linearly for both the datasets. For second experiment, mentioned in Fig. 13b, we incremented the values of sub-sequence length and it can be seen that computational time remains almost linear with increasing the subsequence length. In case of the third experiment, mentioned in Fig. 13c, it can be seen that the computational time to compute kNN MP increases with increasing time series length. Please note that for of these experiments to measure computational time in different scenarios i.e. variation of kNN, time series length, subsequence length are performed by concatenating the individual time series, where we choose less number of individual time series (i.e. approx. 100) contrary to the experiments, mentioned in Section 6.3 (where we choose approx. 1000 time series). The primary objective of these experiments in this section are to show the comparative performance of the proposed algorithm with respect to the other state-of-the art techniques such as rival algorithm such as *STOMP* [2], *SCRIMP* and *SCRIMP++* [2]. That's why, we choose comparatively a smaller dataset (to obtain faster results) to run it on a single core of the computer. Based on the computational complexity of these algorithms (i.e. the proposed algorithm, *STOMP*, *SCRIMP* and *SCRIMP++* algorithms) and the experiments shown in Fig. 13, it is evident that the computational time will increment proportionally in case of bigger time series.

From the first experiment on increasing k nearest neighbors v/s computational time (see Fig. 13a), it can be seen that for seismic dataset, the proposed method has outperformed other techniques, whereas $SCRIMP$ & $SCRIMP{+}{+}$ performed almost equally (although $SCRIMP$ has performed a little better than $SCRIMP{+}{+}$) and has appeared to be the 2^{nd} and 3^{rd} best respectively. The $STOMP$ algorithm took highest time than all other techniques. A very similar performance can be seen for random walk dataset, where the proposed technique has outperformed all other algorithms and $SCRIMP$ $SCRIMP++$, $STOMP$ algorithms has performed at 2^{nd} and 3^{rd} and 4^{th} places respectively. We can see very similar performance in the case of second experiment (see Fig. 13b) on increasing sub-sequence length v/s computational time. Here also, the proposed technique has outperformed other methods on both the datasets. Whereas, the $SCRIMP$ & $SCRIMP++$ algorithms has performed almost equally on both the datasets and has achieved better results than $STOMP$ algorithm. In the third experiment on increasing time series length v/s computational time (see Fig. 13c), it can be seen that the $SCRIMP$ & $SCRIMP++$ has performed slightly better than the proposed algorithm on seismic dataset. Whereas, in the case of random walk dataset, the proposed algorithm has performed better than all other techniques and $SCRIMP$ & $SCRIMP++$ algorithms has performed almost equally in this dataset. Furthermore, in this experiment also the $STOMP$ algorithm took highest computational time in both the datasets.

Hence, it can be visible from these above mentioned experiments that not only the proposed algorithm is able to compute actual kNN MP (contrary to other rival algorithms i.e. $SCRIMP$, $SCRIMP++$ and $STOMP$, which are only able to compute $1NN$ MP in it's original form and has been adapted here to estimate the computational time to compute kNN MP) but it is also faster than other rival algorithms. As mentioned before in this section that the adapted form of these rival techniques are not actually providing the correct kNN MP and these ones are simply adapted to estimate the comparative computational time.

SM: III kNN similarity search for shapelet discovery

In the context of time series, the shapelets can be defined as the sub-sequences which can maximally represent the class of a time series. Recently, the shapelets have been highly used for creating classifiers in various domains such as gesture recognition, sensor networks, motion capture, robotics, electrical power demand etc [18].

Definition 8. ***1NN Similarity join set*** *:* Given all sub-sequences of time series T_A and T_B, the $1NN$ similarity join set J_{AB} of T_A and T_B can be defined as a set containing pairs of each sub-sequences in T_A with its nearest neighbor in T_B. We can formally denote this equation as : $J_{AB} = T_A \bowtie_{\theta\, 1NN} T_B$. We measure the euclidean distance between each pair within a similarity join set and store it into an ordered vector which is called as "matrix profile".

In the context of MP, the authors in [2] have also used STOMP for shapelet discovery. The idea is to consider two time series T_A and T_B, having class 1 and 0 as

their corresponding class labels. Then J_{AA}, J_{AB}, J_{BB} and J_{BA} is calculated for finding the shapelets by computing the difference in heights of P_{AB} v/s P_{AA} (or P_{BA} v/s P_{BB}) which can be used as the indicators of good shapelet candidates. The idea here is that if a discriminating pattern is present in T_A and this particular pattern is not present at all in T_B then it is highly probable that we will see a "bump" at the location of this pattern in P_{AB} (the same is true for P_{BA} also). Hence, when an element-wise difference will be calculated between P_{AA} and P_{AB} vectors, we will find high values at those locations where such discriminating patterns (or sub-sequences) exists in T_A (same is true for T_B, if we look into P_{BB} and P_{BA}).

If some value is low in P_{AA} and at the same location it is high in P_{AB} curve then it means that the particular sub-sequence exists (say "query sub-sequence") at this location has found one close match ($1NN$) within the time series T_A. But the best match coming from T_B of the same "query sub-sequence", is not very close and that's why it shows a high distance in P_{AB}. It may also be possible that some value in P_{AA} curve is lower than the value exists at the same location of P_{AB} curve (i.e. $P_{AA} < P_{AB}$). It means that the best match of the "query sub-sequence" (coming from T_A) is not better than the best match coming from T_B. Hence, if we compute $P_{AA} - P_{AB}$ then we will get a negative value at this location/index of the "query sub-sequence" and this kind of sub-sequences/patterns can't be a good representative of T_A as it's best match which is coming from T_A is worse than it's best match which is coming from T_B. So, we ignore these kind of sub-sequences to be chosen as shapelets.

The occurrence of low value in P_{AA} is also possible due to some presence of noise/outliers (see the case explained in Fig. 7). So, computing $1NN$ matrix profile of T_A (i.e. $P_{AA}(1NN)$) is not sufficient to be sure that this particular "query sub-sequence" is a representative pattern in T_A. The computation of $1NN$ matrix profile confirms the presence of only one nearest neighbor in the time series. For that reason, we need to compute $2NN$, $3NN$, $4NN$ ... kNN etc (k is an user given parameter). In Fig. 14a, we have plotted the curve of P_{AA} and P_{AB} (curve of P_{BB} and P_{BA} is plotted in Fig. 14b). The difference between $P_{AA}(1NN)$ and $P_{AB}(1NN)$ (named as $Diff_1$), along with the difference between $P_{AA}(2NN)$ and $P_{AB}(1NN)$ (named as $Diff_2$) are plotted in Fig. 14c (the threshold is marked by dotted blue line). The values in $Diff_2$ is lesser than $Diff_1$ (i.e. $Diff_2 < Diff_1$) because the values in $P_{AA}(2NN)$ is greater than $P_{AA}(1NN)$ (i.e. $P_{AA}(2NN) > P_{AA}(1NN)$). The same logic is applicable for P_{BB} and P_{BA} (see Fig. 14d). Hence, if the value is small in $P_{AA}(1NN)$ and it remain small in $P_{AA}(2NN)$ also (where $P_{AA}(2NN) > P_{AA}(1NN)$), then we can say that the "query sub-sequence" has not only one close match, it also has a 2^{nd} close match. Whereas if we find a high value at the same index in $P_{AB}(1NN)$, then it can be said that the "query sub-sequence of T_A could not find a good match in T_B. Hence by computing $P_{AA}(2NN)$, we can be further sure that the chosen "query sub-sequence" of T_A is a good representative of T_A. The same logic can be further applied for $P_{AA}(4NN)$ (shown in Fig. 14e) and $P_{BB}(4NN)$ also (shown in Fig. 14f) for further being sure about the choice of shapelets or patterns from T_A (or T_B).

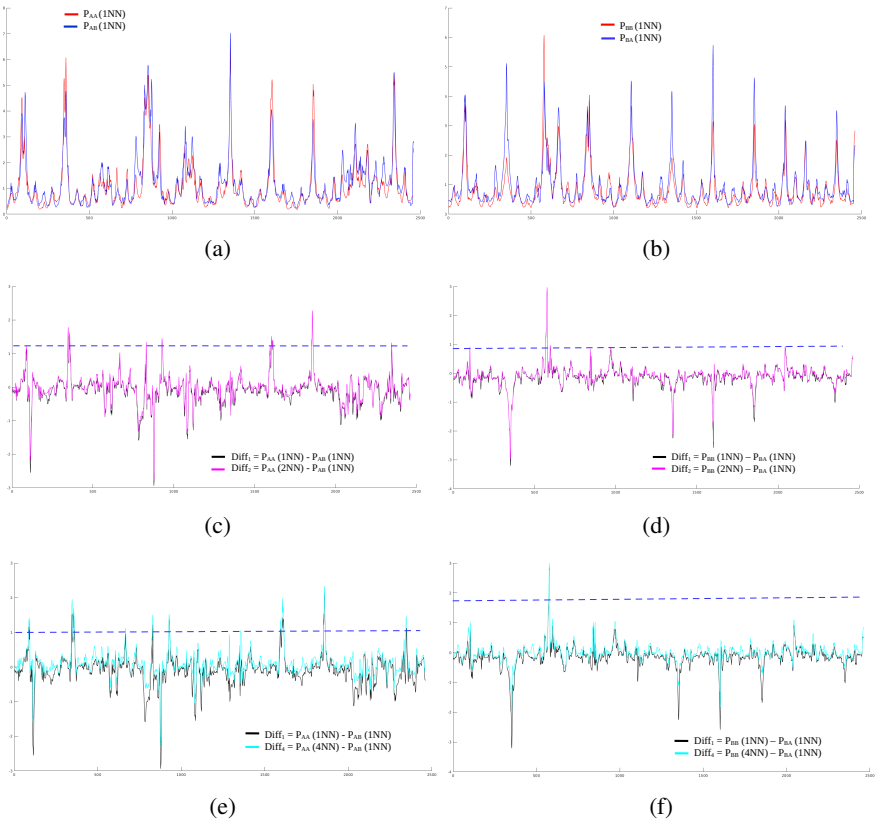


Figure 14: (a) The matrix profile $P_{AA}(1NN)$ and $P_{AB}(1NN)$ is plotted. (b) The matrix profile $P_{BB}(1NN)$ and $P_{BA}(1NN)$ is plotted. (c) The difference between $P_{AA}(1NN)$ v/s $P_{AB}(1NN)$ is plotted with black color and the difference between $P_{AA}(2NN)$ v/s $P_{AB}(1NN)$ is plotted with pink color. (d) The difference between $P_{BB}(1NN)$ v/s $P_{BA}(1NN)$ is plotted with black color and the difference between $P_{BB}(2NN)$ v/s $P_{BA}(1NN)$ is plotted with pink color. (e) The difference between $P_{BB}(1NN)$ v/s $P_{BA}(1NN)$ and the difference between $P_{BB}(4NN)$ v/s $P_{BA}(1NN)$ are plotted. (f) The difference between $P_{BB}(1NN)$ v/s $P_{BA}(1NN)$ and the difference between $P_{BB}(4NN)$ v/s $P_{BA}(1NN)$ are plotted.

For the experiments, we have randomly selected 10 time series (like in [2]) of class 0 and 1 from the test set of "Arrow Head dataset" [19]. Each of these time series is of length 251 and by concatenating 10 time series of class 0 and 1, we obtain two big time series of length 2510 of class 0 and 1 respectively. The computational time required to perform P_{AA} and P_{AB} (for time series T_A and T_B of length 2510) are 0.93 and 0.91 seconds respectively (no multi-core based parallel computation is used) on an off-the-shelf computer.

SM: IV Outliers and/or motifs detection by using kNN MP

In continuation with the experiments, mentioned in Section 6.2.3, here we talk about some extended experiments of kNN MP by using the datasets from UCR time series data mining archive. To create a single big time series from each dataset of UCR archive, we have sequentially concatenated the individual time series from training and testing set. Let's consider an example where in any particular time series dataset $\mathfrak{D}^T$ contains $\mathcal{X}$ number of individual time series of same lengths : $\{T_1, T_2, T_3, ..., T_{\mathcal{X}}\} \in \mathfrak{D}^T$. We choose any one time series T_q as query and the remaining time series from $\mathfrak{D}^T$ are concatenated to generate a big time series T^{concat} which is named as *target time series*. Now the objective is to match two independent time series i.e. T_q and T^{concat} by finding the k nearest neighbors of all the subsequence in T_q and the matches would come from T^{concat} only. That's how, we can generate kNN MP of T_q time series by using the Algorithm 1.

In this experiment, we choose 30 query time series[7] from each dataset and for each chosen query time series, we follow the above mentioned procedure to generate $1NN$ (denoted as $P_{T_q}^{1NN}$), $2NN$ (denoted as $P_{T_q}^{2NN}$) and $4NN$ (denoted as $P_{T_q}^{4NN}$) MP for the particular query time series. Then we compute the difference between $1NN$ v/s $2NN$ MPs and $1NN$ v/s $4NN$ MPs in the following manner :

$$\mathcal{D}_{diff}^{1-2} = |P_{T_q}^{2NN} - P_{T_q}^{1NN}| \tag{4}$$

$$\mathcal{D}_{diff}^{1-4} = |P_{T_q}^{4NN} - P_{T_q}^{1NN}| \tag{5}$$

Now, for each query time series T_q, we choose top 20 high values from $\mathcal{D}_{diff}^{1-2}$ and $\mathcal{D}_{diff}^{1-4}$ respectively. The objective here is to find the points where there exists high difference between $P_{T_q}^{2NN}$ v/s $P_{T_q}^{1NN}$ and $P_{T_q}^{4NN}$ v/s $P_{T_q}^{1NN}$; which will finally help us to locate the points where kNN (i.e. $2NN$ and $4NN$) MP has shown different characteristics than $1NN$ MP. In this manner, we can obtain $30 \times 20 = 600$ values from $\mathcal{D}_{diff}^{1-2}$ and another 600 values from $\mathcal{D}_{diff}^{1-4}$ for each UCR dataset. We also compute the maximum value (ξ_{2NN}) and (ξ_{4NN}) from $P_{T_q}^{2NN}$ and $P_{T_q}^{4NN}$ over all the 30 query time series. The ξ_{2NN} and ξ_{4NN} values are used for dividing the chosen 600 values from $\mathcal{D}_{diff}^{1-2}$ and 600 values from $\mathcal{D}_{diff}^{1-4}$ respectively for the purpose of normalization. Let's denote these normalized values as γ_{2NN}^{600} and $\gamma_4^{600}NN$ respectively. Then, three different threshold values are chosen by taking 2%, 5% and 10% of ξ_{2NN} (named as $\tau_{2NN}^{2\%}$, $\tau_{2NN}^{5\%}$, $\tau_{2NN}^{10\%}$) and ξ_{4NN} (named as $\tau_{4NN}^{2\%}$, $\tau_{4NN}^{5\%}$, $\tau_{4NN}^{10\%}$) respectively. Then based on these thresholds i.e. $\tau_{2NN}^{2\%}$, $\tau_{2NN}^{5\%}$, $\tau_{2NN}^{10\%}$, we count the total number of values in γ_{2NN}^{600} which pass these thresholds. The same operation is also performed for the case of γ_{4NN}^{600} and the detailed statistics of 87 UCR datasets are noted in Table S2 and Table S3. These statistics are also represented in a histogram format in Fig. 15 and Fig. 16. In can be visible that with the increase of threshold value the count for γ_{2NN}^{600} and γ_{4NN}^{600} decreases. These counts, shown in Fig. 15 and Fig. 16 represents the number of points in $2NN$ and $4NN$ MP has strong difference

[7] See "UCR_Matrix_Profile_Outliers_Motifs.m" in https://github.com/tanmayGIT/kNN_Matrix_Profile.git

than their counterpart in $1NN$ MP and these characteristics could not be analyzed if we wouldn't have computed kNN MP.

We have also shown several more visual examples of kNN MP in Fig. 17, Fig. 18, Fig. 19, Fig. 20, Fig. 21. It can be visible that at several points (i.e. at the matrix profile indexes) there exists strong difference between $1NN$ v/s $2NN$ MP and $1NN$ v/s $4NN$ MP. These kNN MP plots helps use to understand and analyze the following cases :

1. There are cases, where the $1NN$ MP will identify any particular point as *motif* (based on user defined threshold) but either $2NN$ or $4NN$ MP curve wouldn't show the same characteristic (may even show opposite form i.e. $1NN$ is downward but $2NN$ or $4NN$ is upward) at the same point. In such cases, the detected *motif* by $1NN$ MP should be considered with precaution.
2. There are cases, where any particular point in $1NN$ MP is not detected as *discord* (based on user defined threshold) but either $2NN$ or $4NN$ MP curve show completely different characteristic (may even show opposite form i.e. $1NN$ is downward but $2NN$ or $4NN$ is upward) at the same point and got detected as discord (due to high value at it's corresponding MP). In such cases, the detected *discord* by $2NN$ or $4NN$ MP should be considered with precaution also.

We haven't explicitly drawn the user defined threshold lines for the detection of *motifs* and/or *discords* in all of these curves (as we did in Fig. 9 and Fig. 8) because the choice of threshold is user dependent and the primary purpose to show all of these figures are to visualize the importance of kNN MP over $1NN$ MP which is clearly served from these illustrations.

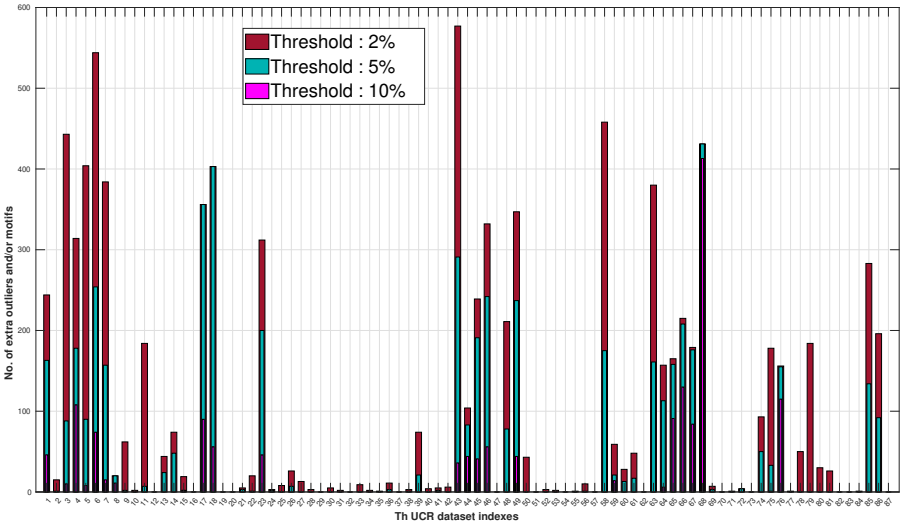


Figure 15: The histogram of the difference between $1NN$ v/s $2NN$ MPs, denoted as $\mathcal{D}_{diff}^{1-2}$. Three threshold values i.e. $\tau_{2NN}^{2\%}$, $\tau_{2NN}^{5\%}$, $\tau_{2NN}^{10\%}$ are used to get statistics. These stats, corresponding to $\tau_{2NN}^{2\%}$, $\tau_{2NN}^{5\%}$, $\tau_{2NN}^{10\%}$ are plotted in violet, green and pink colors here. For more details about these stats, please see Section SM: IV, Table S2 and Table S3

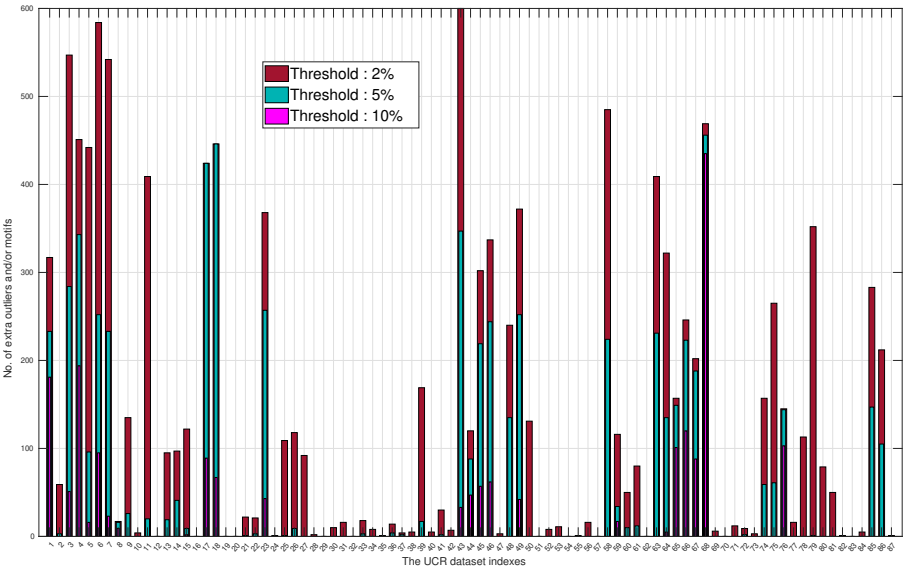


Figure 16: The histogram of the difference between $1NN$ v/s $4NN$ MPs, denoted as $\mathcal{D}_{diff}^{1-4}$. Three threshold values i.e. $\tau_{4NN}^{2\%}$, $\tau_{4NN}^{5\%}$, $\tau_{4NN}^{10\%}$ are used to get statistics. These stats, corresponding to $\tau_{4NN}^{2\%}$, $\tau_{4NN}^{5\%}$, $\tau_{4NN}^{10\%}$ are plotted in violet, green and pink colors here. For more details about these stats, please see Section SM: IV, Table S2 and Table S3

Table S2: Dataset details and the statistics of the difference between $1NN$ v/s $2NN$ & $1NN$ v/s $4NN$ from UCR archive. For more details, see Section SM: IV

SL. No.	Dataset Name	Size of training set	Size of testing set	Time series length	1NN v/s 2NN			1NN v/s 4NN		
					2%	5%	10%	2%	5%	10%
1	ACSF1	100	100	1461	244	163	46	317	233	181
2	Adiac	390	391	177	15	1	0	59	3	0
3	ArrowHead	36	175	252	443	88	10	547	284	51
4	BME	30	150	129	314	178	108	451	343	194
5	Beef	30	30	471	404	90	8	442	96	16
6	BeetleFly	20	20	513	544	254	74	584	252	95
7	BirdChicken	20	20	513	384	157	15	542	233	23
8	CBF	30	900	129	20	20	11	17	16	9
9	Car	60	60	578	62	2	0	135	26	0
10	ChlorineConcentration	467	3840	167	2	0	0	4	0	0
11	CinCECGTorso	40	1380	1640	184	7	0	409	20	0
12	Coffee	28	28	287	0	0	0	0	0	0
13	CricketX	390	390	301	44	24	0	95	19	0
14	CricketY	390	390	301	74	48	0	97	41	0
15	CricketZ	390	390	301	19	2	2	122	9	2
16	DiatomSizeReduction	16	306	346	0	0	0	0	0	0
17	DodgerLoopDay	78	80	289	356	356	90	424	424	89
18	DodgerLoopWeekend	20	138	289	403	403	56	446	446	67
19	ECG5000	500	4500	141	0	0	0	0	0	0
20	ECGFiveDays	23	861	137	0	0	0	0	0	0
21	EOGHorizontalSignal	362	362	1251	5	3	0	22	1	0
22	EOGVerticalSignal	362	362	1251	20	1	0	21	3	0
23	Earthquakes	322	139	513	312	200	46	368	257	43
24	EthanolLevel	504	500	1752	3	2	0	1	1	0
25	FaceAll	560	1690	132	8	0	0	109	1	0
26	FaceFour	24	88	351	26	7	0	118	9	0
27	FacesUCR	200	2050	132	13	0	0	92	0	0
28	FiftyWords	450	455	271	3	0	0	2	0	0
29	Fish	175	175	464	0	0	0	0	0	0
30	FordA	3601	1320	501	5	0	0	10	0	0
31	FordB	3636	810	501	2	0	0	16	0	0
32	FreezerRegularTrain	150	2850	302	0	0	0	0	0	0
33	FreezerSmallTrain	28	2850	302	9	0	0	18	3	0
34	Fungi	18	186	202	2	0	0	8	0	0
35	GunPoint	50	150	151	1	0	0	1	1	0
36	GunPointAgeSpan	135	316	151	11	3	0	14	4	0
37	GunPointMaleVersusFemale	135	316	151	0	0	0	4	2	0
38	GunPointOldVersusYoung	136	315	151	3	1	0	5	0	0
39	Ham	109	105	432	74	21	0	169	17	0
40	HandOutlines	1000	370	2710	4	0	0	5	0	0
41	Haptics	155	308	1093	5	2	0	30	2	0
42	Herring	64	64	513	6	0	0	7	0	0
43	HouseTwenty	40	119	2001	577	291	36	600	347	33
44	InlineSkate	100	550	1883	104	83	44	120	88	47
45	InsectEPGRegularTrain	62	249	602	239	191	41	302	219	57
46	InsectEPGSmallTrain	17	249	602	332	242	56	337	244	62
47	InsectWingbeatSound	220	1980	257	0	0	0	3	0	0
48	LargeKitchenAppliances	375	375	721	211	78	0	240	135	0
49	Lightning7	70	73	320	347	237	44	372	252	42
50	Mallat	55	2345	1025	43	0	0	131	0	0

Table S3: Dataset details and the statistics of the difference between $1NN$ v/s $2NN$ & $1NN$ v/s $4NN$ from UCR archive. For more details, see Section SM: IV

					1NN v/s 2NN			1NN v/s 4NN		
SL. No.	**Dataset Name**	**Size of training set**	**Size of testing set**	**Time series length**	**2%**	**5%**	**10%**	**2%**	**5%**	**10%**
51	Meat	60	60	449	0	0	0	0	0	0
52	MixedShapesRegularTrain	500	2425	1025	3	0	0	8	0	0
53	MixedShapesSmallTrain	100	2425	1025	2	0	0	11	0	0
54	NonInvasiveFetalECGThorax1	1800	1965	751	0	0	0	0	0	0
55	NonInvasiveFetalECGThorax2	1800	1965	751	1	0	0	1	0	0
56	OSULeaf	200	242	428	10	1	0	16	0	0
57	OliveOil	30	30	571	0	0	0	0	0	0
58	Phoneme	214	1896	1025	458	175	0	485	224	0
59	PigAirwayPressure	104	208	2001	59	21	14	116	34	17
60	PigArtPressure	104	208	2001	28	13	0	50	10	0
61	PigCVP	104	208	2001	48	17	0	80	12	0
62	Plane	105	105	145	0	0	0	0	0	0
63	PowerCons	180	180	145	380	161	0	409	231	0
64	Rock	20	50	2845	157	113	6	322	135	5
65	SemgHandGenderCh2	300	600	1501	165	158	91	157	149	101
66	SemgHandMovementCh2	450	450	1501	215	208	130	246	223	120
67	SemgHandSubjectCh2	450	450	1501	179	176	84	202	188	88
68	ShapeletSim	20	180	501	431	431	413	469	456	435
69	ShapesAll	600	600	513	7	3	0	6	0	0
70	StarLightCurves	1000	8236	1025	0	0	0	0	0	0
71	Strawberry	613	370	236	1	0	0	12	0	0
72	SwedishLeaf	500	625	129	4	4	0	9	2	0
73	Symbols	25	995	399	0	0	0	3	0	0
74	ToeSegmentation1	40	228	278	93	50	0	157	59	0
75	ToeSegmentation2	36	130	344	178	33	1	265	61	0
76	Trace	100	100	276	156	155	115	145	144	103
77	TwoPatterns	1000	4000	129	1	0	0	16	0	0
78	UMD	36	144	151	50	0	0	113	0	0
79	UWaveGestureLibraryAll	896	3582	946	184	0	0	352	1	0
80	UWaveGestureLibraryX	896	3582	316	30	0	0	79	0	0
81	UWaveGestureLibraryZ	896	3582	316	26	0	0	50	0	0
82	Wafer	1000	6164	153	0	0	0	1	0	0
83	Wine	57	54	235	0	0	0	0	0	0
84	WordSynonyms	267	638	271	1	0	0	5	0	0
85	Worms	181	77	901	283	134	0	283	147	0
86	WormsTwoClass	181	77	901	196	92	0	212	105	0
87	Yoga	300	3000	427	0	0	0	1	0	0

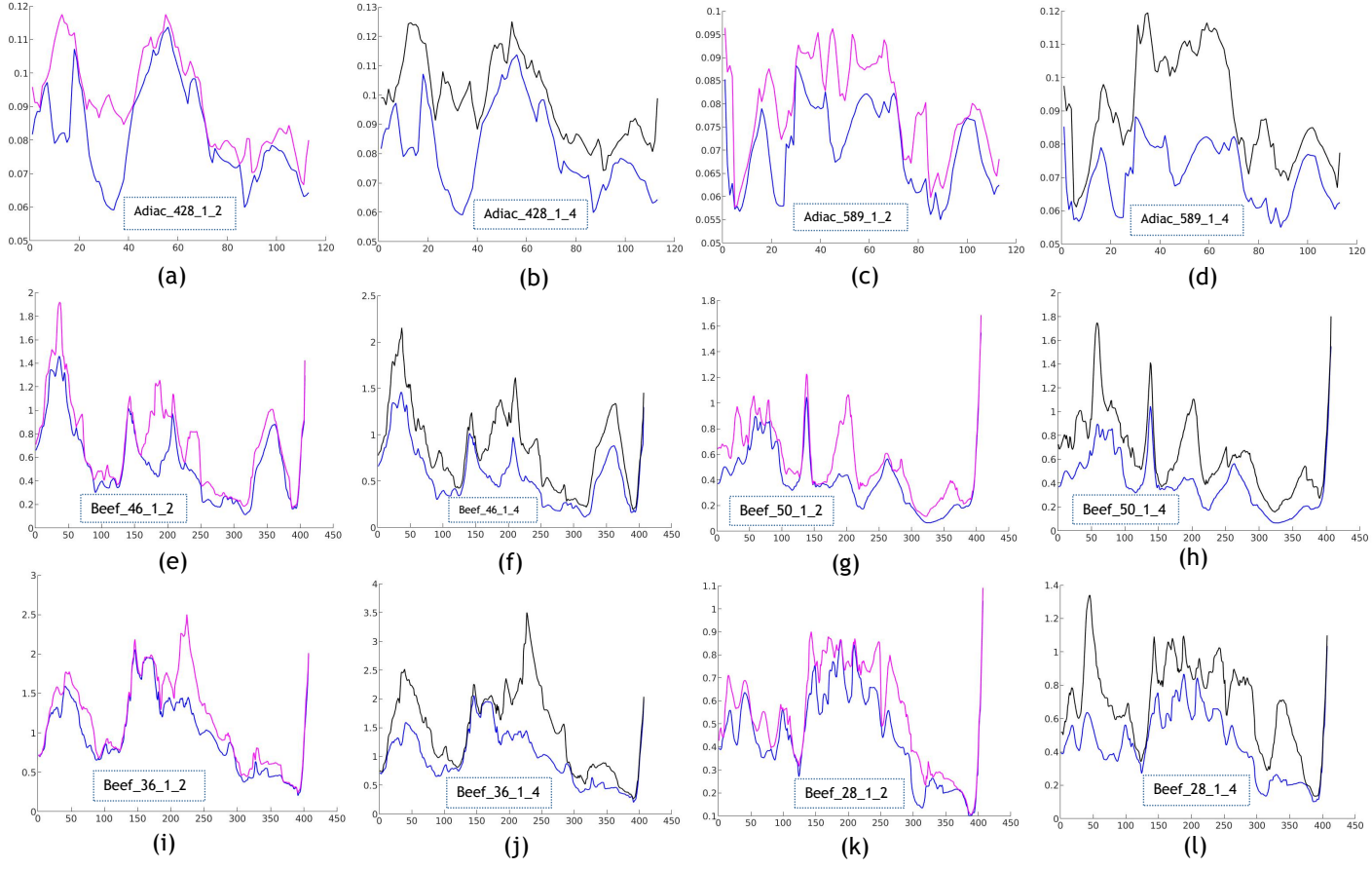


Figure 17: The results of kNN MP detection from few selected UCR dataset. The legends, mentioned in the curves can be interpreted as: "nm_idx_1NN_kNN"; where nm = the dataset name; idx = the index of query time series; 1NN_kNN = the 1NN v/s kNN plots, where $k = 2, 4$.

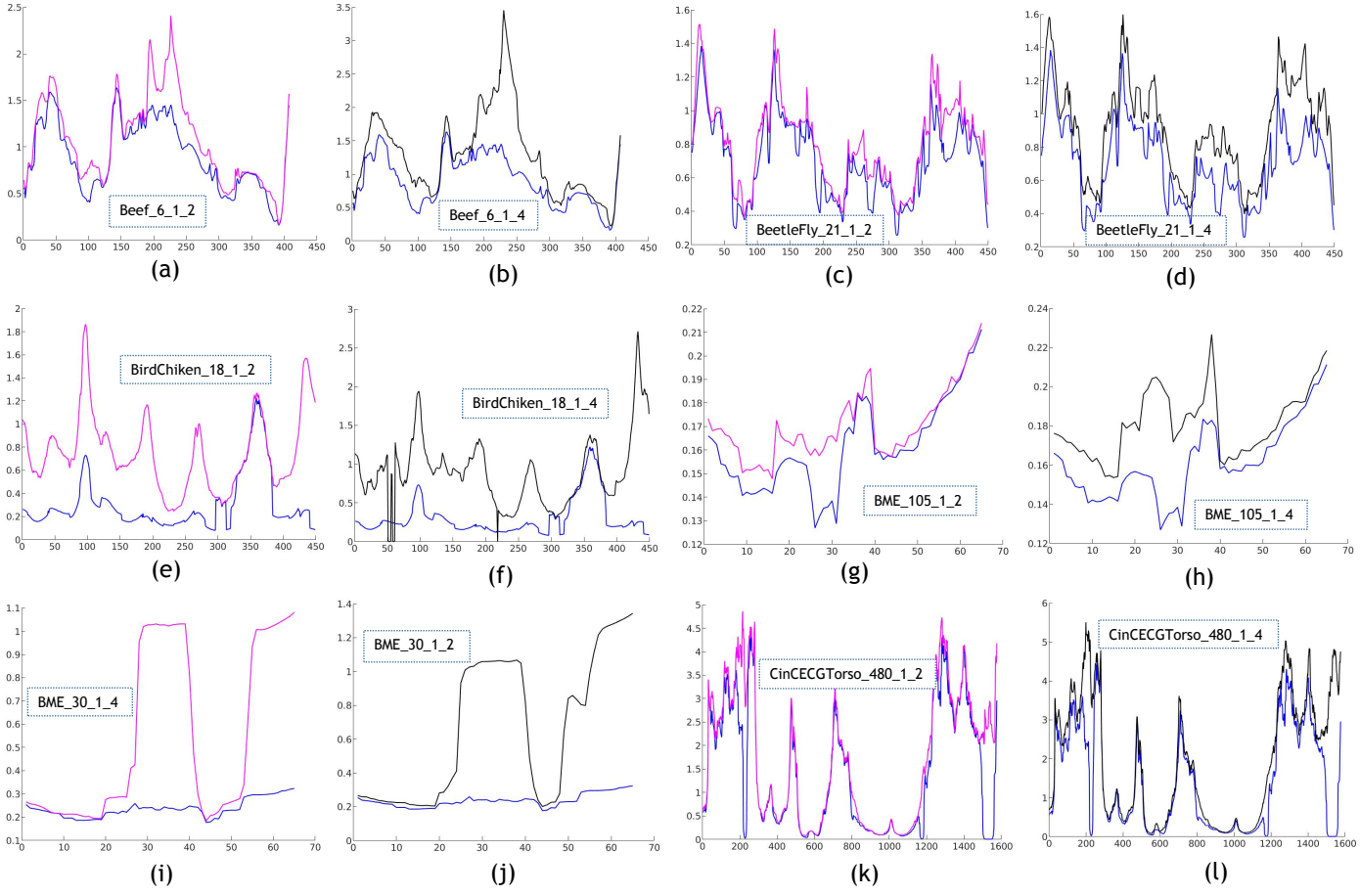


Figure 18: The results of kNN MP detection from few selected UCR dataset. The legends, mentioned in the curves can be interpreted as: "nm_idx_1NN_kNN"; where nm = the dataset name; idx = the index of query time series; 1NN_kNN = the 1NN v/s kNN plots, where $k = 2, 4$.

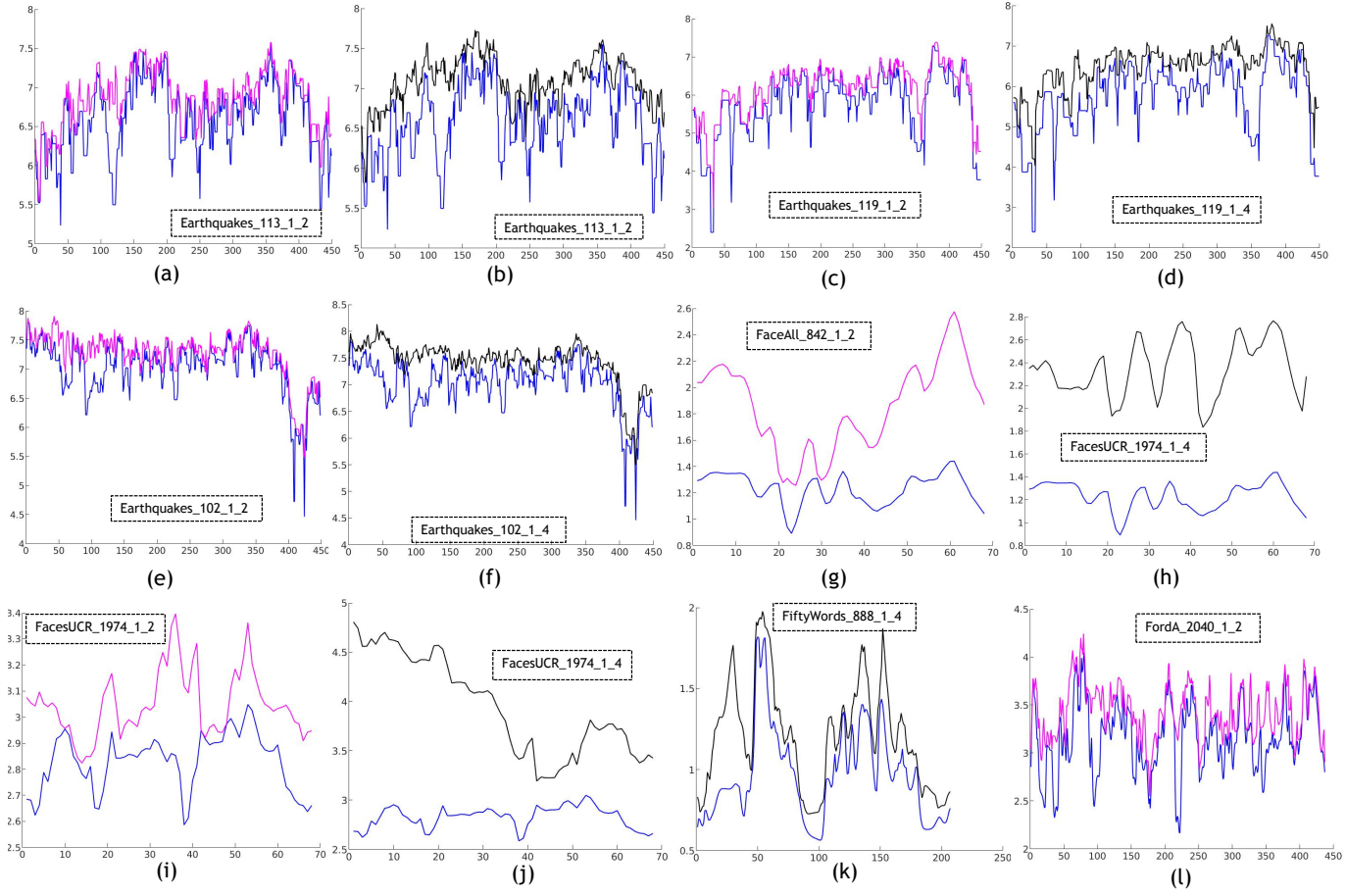


Figure 19: The results of kNN MP detection from few selected UCR dataset. The legends, mentioned in the curves can be interpreted as: "nm_idx_1NN_kNN"; where nm = the dataset name; idx = the index of query time series; 1NN_kNN = the 1NN v/s kNN plots, where $k = 2, 4$.

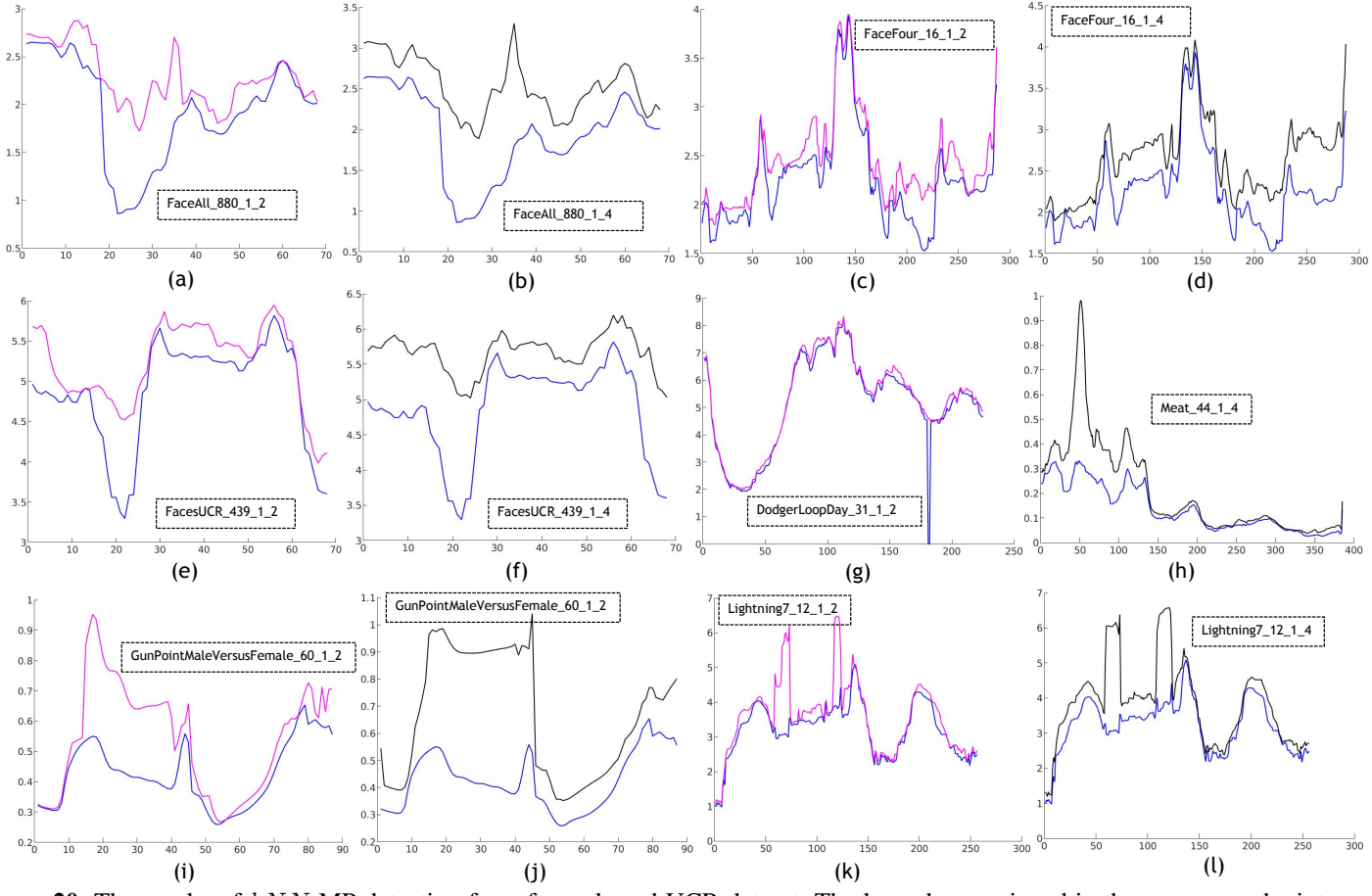


Figure 20: The results of kNN MP detection from few selected UCR dataset. The legends, mentioned in the curves can be interpreted as: "nm_idx_1NN_kNN"; where nm = the dataset name; idx = the index of query time series; 1NN_kNN = the 1NN v/s kNN plots, where $k = 2, 4$.

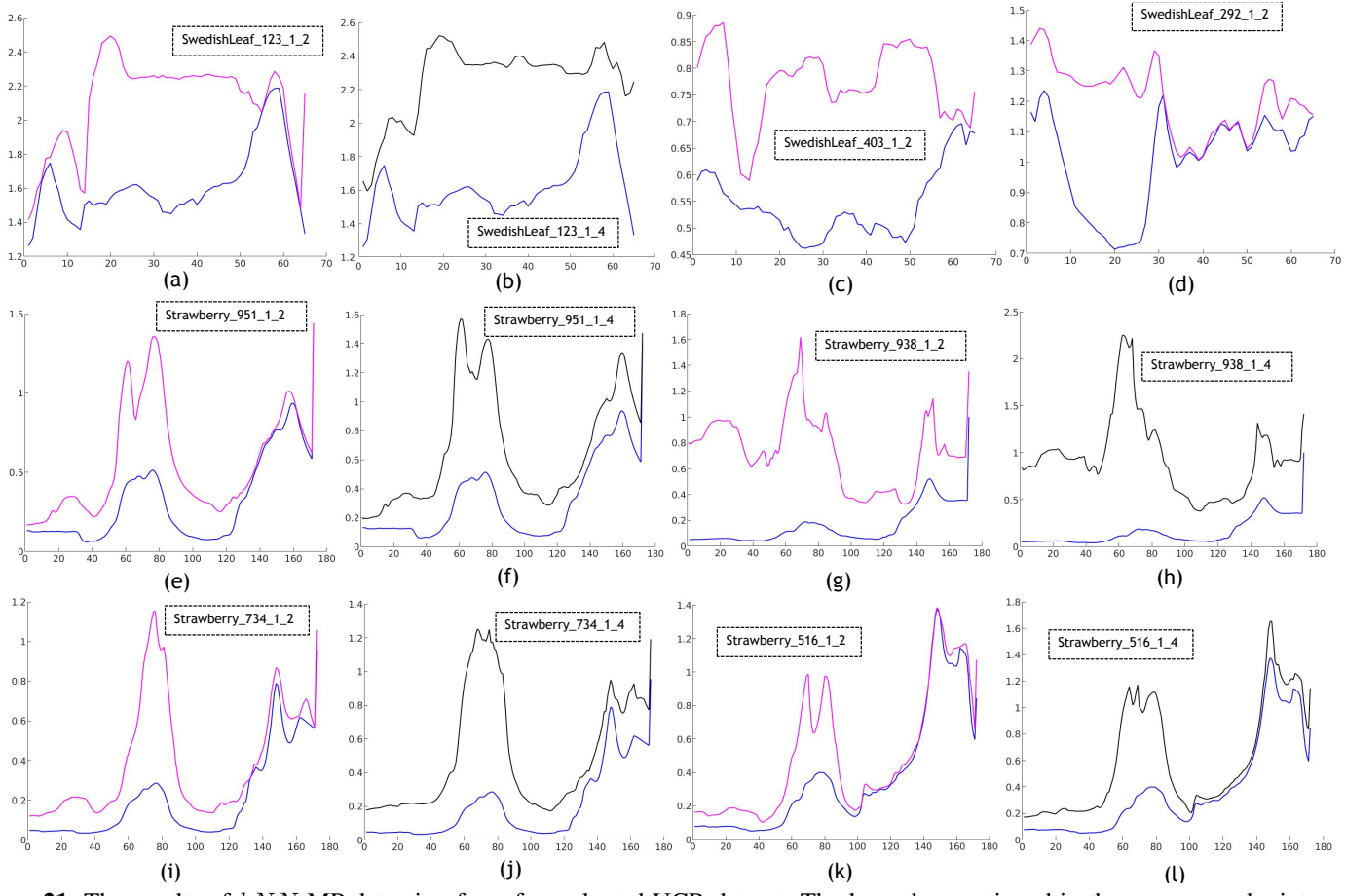


Figure 21: The results of kNN MP detection from few selected UCR dataset. The legends, mentioned in the curves can be interpreted as: "nm_idx_1NN_kNN"; where nm = the dataset name; idx = the index of query time series; 1NN_kNN = the 1NN v/s kNN plots, where $k = 2, 4$.