MASS: Distance Profile of a Query over a Time Series

Sheng Zhong[1] and Abdullah Mueen[2]

[1,2]Department of Computer Science, The University of New Mexico, Albuquerque, 87106, New Mexico, USA.

Contributing authors: zhongs@unm.edu; mueen@unm.edu;

Abstract

Given a long time series, the distance profile of a query time series computes distances between the query and every possible subsequence of a long time series. MASS (Mueen's Algorithm for Similarity Search) is an algorithm to efficiently compute distance profile under z-normalized Euclidean distance [1]. MASS is recognized as a useful tool in many data mining works. However, complete documentation of the increasingly efficient versions of the algorithm does not exist.

In this paper, we formalize the notion of a distance profile, describe four versions of the MASS algorithm, show several extensions of distance profiles under various operating conditions, describe how MASS improves performances of existing data mining algorithms, and finally, show utility of MASS in domains including seismology, robotics and power grids.

Keywords: Time Series, Distance Profile, Correlation, Euclidean Distance, Convolution

1 Introduction

Time series is a sequence of observations made in time order. Given a query time series, the similarities or distances of the query to all possible subsequences of a time series constitute a distance profile of the query. Computing distance profile is a fundamental task in time series data mining and has been utilized in many existing works [2][3][4]. For example, to compute the Matrix Profile of a time series, the STAMP algorithm [5] repeatedly computes the distance profiles of subsequences of the given series. However, even though a rising interest in profiling time series data [6] has been

observed, the literature does not present a formal and comprehensive understanding of the algorithms for distance profiling of a query over a time series.

In this paper, we define the time series distance profile under various operating requirements and provide detailed discussions on the algorithms, extensions, utility and use cases. We describe four algorithms to compute the distance profile under Euclidean distance. The algorithms are incrementally efficient and uniquely useful. We describe four different extensions of the distance profile: weighted Euclidean distance profile, un-normalized Euclidean distance profile, correlation profile and partial correlation profile for dual queries. We present faster algorithms for time series discord discovery and time series subsequence clustering exploiting distance profiles than the traditional search-and-prune algorithms. We finally show three novel use cases of distance profiles in the domains of seismology, power consumption, and robotics.

2 Related Work

The distance profile serves as a foundational element in time series data mining and machine learning. It measures the similarities between a query sequence and subsequences within a longer time series. This measurement is integral to a range of critical tasks, including motif discovery, anomaly detection, and classification. MASS significantly boosts the efficiency of distance profile computation without compromising the precision of the results. The subsequent sections will provide an overview of the wide-ranging applications of the MASS algorithm, illustrating its significance in diverse data mining and machine learning tasks.

Pattern recognition: A fundamental application of the MASS algorithm lies in its ability to identify specific patterns that are related to certain events or characteristics. An example of this is its use in recognizing the actions of electrical appliances within a household by computing distance profiles of unique power consumption patterns against smart meter measurements [7]. The authors claim that this approach boosted by MASS can be carried out in (near) real-time using edge computing. Additionally, MASS can also help with real-time defect detection during metal additive manufacturing [8].

Matrix profile calculation for motif discovery: When identifying motifs without the knowledge of the specific pattern to query, the matrix profile (MP) [5] becomes an invaluable tool. The construction of MP involves determining the distance profile for every possible subsequence within a time series. Here, the MASS algorithm is crucial, serving as a key component in efficient MP calculation. It is integrated into algorithms such as STAMP [5] and SCRIMP++ [9], where it significantly enhances their performance. Although not directly incorporated in STOMP [10], the optimization methods used for calculating dot products in MASS versions 3 and 4 are still influential and relevant.

Domain-specific variations of matrix profile: Beyond the standard matrix profile, the MASS algorithm facilitates the development of specialized profiles tailored to specific fields of study. Notable examples include: similarity matrix profile (SiMPle) for cover song identification [11, 12], this variant leverages MASS to analyze and compare musical pieces, demonstrating its effectiveness in the field of musicology. In-phase

matrix profile applied in EEG data analysis [13]. Radius profile employed for identifying repeating subsequences useful in various analytical, MASS aids in recognizing patterns that recur within time series. Analog ensemble profile used in meteorological analysis [14], MASS contributes to the creation of analog ensembles, enhancing the accuracy and depth of weather-related predictions and studies.

Classification and clustering: Many time series classification and clustering methods that rely on the distance profile can benefit from MASS. For instance, Abdoli et al. leveraged MASS to classify chicken behaviors [15]. They utilized a labeled subsequence representing a specific behavior as a query. MASS computed the distance profile against a streaming time series of chicken movements within a certain period. Subsequences similar to the query were classified accordingly. Heo et al. demonstrated the application of MASS in music. They used the algorithm to calculate the distance profile for individual songs in a test set against a composite time series created by concatenating all songs from a training set [16]. Lin et al. developed improved embeddings for classification tasks based on the distance profiles computed by MASS, showcasing its potential in refining data representation for better classification accuracy [17]. Emerging research continues to explore the capabilities of MASS in handling vast volumes of time series data for classification, One study [17] demonstrated MASS enables classification on time series with billions of samples, underscoring its scalability and efficiency.

Prediction: A key use of MASS in prediction involves comparing recent observations with historical data to identify similar past scenarios and use them to forecast near-future states. This approach is particularly effective when a substantial amount of historical data is available, which inherently demands more computational power. MASS demonstrates its significant utility in this process, for instance, MASS was employed to compare current weather data against a dataset containing the past 20 years' historical observations for solar power prediction [14]. Further research indicates that MASS can enhance resolution by up to 400%, transforming hourly forecasts into more precise 15-minute intervals [18] which is crucial for applications requiring high-resolution data and timely decision-making. The accuracy can be further improved by incorporating data from various sources, such as a sensor network. MASS's adaptability to different data dimensions makes it well-suited for analyzing complex, multi-source datasets [19, 20].

Anomaly detection: While prediction with MASS is about forecasting future states based on historical data, its use in anomaly detection serves a different purpose. In this context, the MASS algorithm is adept at identifying patterns that are unprecedented or deviate from the norm. In environments where data is continually updated and immediate response is required, MASS's ability to quickly and accurately identify anomalies is invaluable. MASS also excels in analyzing vast datasets where manual detection of anomalies would be impractical or impossible. Referenced studies [8, 21–24] highlight the above scenarios where MASS has been successfully applied in anomaly detection, demonstrating its versatility and effectiveness in this area.

3 Background

In this section, we define the necessary terms and concepts to describe the algorithms in Section 3.

Definition 1 (time series): A time series T of length n is an ordered sequence of real numbers $T[i]$ measured in equally spaced time, in which $T = (T[1], T[2], \ldots, T[n])$.

Definition 2 (subsequence): A subsequence $T_{i,L}$ is a continuous segment of length L from a time series T starting from position i. $T_{i,L} = (T[i], T[i+1], \ldots, T[i+L-1])$, where $1 \leq i \leq n - L + 1$. For a time series of length n, there can be a total of $\frac{n(n+1)}{2}$ subsequences of all possible lengths.

Definition 3 (query): A time series Q of length m, which is searched within a time series T of length $n >> m$.

Definition 4 (distance profile): Given a time series T and a query Q the distance profile is another sequence D of length $n - m + 1$ such that $D[i] = dist(T_{i,m}, Q)$.

Here, *dist* is a distance function that defines the distance between two equal-length time series. Typical distance functions include Euclidean distance, Pearson's correlation coefficient, cosine similarity and angular distance. Some distance functions can compare two unequal length time series such as dynamic time warping (DTW) [25], longest common subsequence (LCSS) [26] and move-split-merge (MSM) [27]. One can consider more variations of the distance profile by allowing the distance function more flexibility, such as by removing the end-point constraint [28]. However, the definition of distance profile is not limited to how the distance function operates on the pair of time series. The distance functions can also differ in their ranges. Closed ranges increase the utility of the distance profile (as we describe later) by allowing meaningful aggregation operations such as MIN and MAX. The distance function can be discontinuous by generally treating all dissimilar subsequences in the same way by assigning ∞ distance allowing us to speed up computation.

In most parts of this paper, we consider z-normalized Euclidean distance as our distance measure without any discontinuity. The z-normalized Euclidean distance between two time series x and y of length m is defined in Equation 1. Here X is the normalized time series, and x is the original time series.

$$dist(x, y) = \sqrt{\sum_{i=1}^{m} (X[i] - Y[i])^2} \tag{1}$$

where $X[i] = \frac{x[i] - \mu_x}{\sigma_x}$ for $i = 1, 2, \ldots, m$. μ_x is the mean and σ_x is the sample standard deviation of x. This distance function is bounded between zero and $2m$ [29]. A simple set of steps can lead us to the following working formula for z-normalized Euclidean distance [10].

$$dist(x, y) = \sqrt{2m(1 - \frac{\sum_i x[i]y[i] - m\mu_x\mu_y}{m\sigma_x\sigma_y})} \tag{2}$$

The Algorithm 1 describes the brute force way to compute the distance profile. The algorithm scans the time series T once. At each position, the algorithm normalizes the subsequence $T_{i,m}$ and computes the distance to the normalized query. The algorithm

Algorithm 1: BruteForce(T,Q)

Input: A time series T of length n and a query Q of length m
Output: D, the distance profile of Q in T
`/*Index starting at 1 */`
1 $D[1:n-m+1] \leftarrow 0$ `//` D `has length` $n-m+1$
2 $Q \leftarrow$ `zNorm`(Q)
3 **for** $i \leftarrow 1:n-m+1$ **do**
4 $\quad T' \leftarrow$ `zNorm`$(T_{i,m})$
5 $\quad D[i] \leftarrow \sqrt{\sum_{j=1}^{m}(T'[j]-Q[j])^2}$
6 **end**

saves all distances in the array D. The computational complexity of the algorithm is $O(nm)$. Precisely, the algorithm needs $2m$ arithmetic operations at each iteration. To avoid $2m$ operations in each iteration, we can exploit just-in-time normalization [30] that computes and stores two arrays of cumulative sums that can be used to obtain normalized distances. Since we normalize the query before scanning T, the working formula can be further simplified by assigning $\mu_y = 0$ and $\sigma_y = 1$

$$dist(x,y) = \sqrt{2m(1 - \frac{\sum_i x[i]y[i]}{m\sigma_x})} \tag{3}$$

To exploit this simplified formulation, we need to compute dot products over sliding windows and obtain the standard deviation of the sliding window just in time. We use the following working formula for standard deviation.

$$\sigma_x[i] = \sqrt{\frac{1}{m}\sum_{j=i}^{i+m-1} x[j]^2 - (\frac{1}{m}\sum_{j=i}^{i+m-1} x[j])^2} \tag{4}$$

The function `movstd` from Algorithm 2 demonstrate how to compute the σ by visiting each element $T[i]$ once. This is achieved by computing the array of cumulative sums S and an array of cumulative sums of squares, S_2, over the time series T.

Although the Algorithm 2 does not reduce the overall time complexity of Algorithm 1, there is a $2\times$ speedup that we consider as the baseline algorithm that one would consider for computing distance profile.

4 MASS: Mueen's Algorithm for Similarity Search

We describe an $O(n \log n)$ algorithm to compute the distance profile under z-normalized Euclidean distance. We claim the proposed approach is faster than the baseline approach (Algorithm 2) that has a time complexity of $O(nm)$ since $m > \log n$ holds for most real-world applications. The core idea is to use convolution operation. We explain an intuitive example and then formally describe the algorithm.

Algorithm 2: JustInTime(T,Q)

Input: A time series T of length n and a query Q of length m
Output: D, the distance profile of Q in T

$D[1:n-m+1] \leftarrow 0$
$Q \leftarrow \texttt{zNorm}(Q)$
$\sigma_T \leftarrow \texttt{movstd}(T, m)$
for $i \leftarrow 1 : n-m+1$ **do**
  $D[i] \leftarrow \sqrt{2*(m-\sum_{j=1}^{m}(T[i+j-1]*Q[j])/\sigma_T[i])}$
end
Function `movstd`(T, m)
  $N \leftarrow \texttt{length}(T)$
  $S[1:N+1] \leftarrow 0$
  $\sigma[1:N-m+1] \leftarrow 0$
  for $i \leftarrow 2 : N+1$ **do**
    $S[i] \leftarrow S[i-1] + T[i-1]$
  end
  $S_2 \leftarrow S^2$
  for $i \leftarrow m+1 : N+1$ **do**
    $j \leftarrow i-m$
    $\sigma[j] \leftarrow \sqrt{\frac{1}{m}(S_2[i]-S_2[j]) - (\frac{1}{m}(S[i]-S[j]))^2}$
  end
  return σ
end

4.1 MASS V1

We apply two optimizations to achieve $O(n \log n)$. First, we use convolution to compute the dot products in Equation 3. Second, we utilize the properties of the Discrete Fourier Transform to compute the convolution.

If x and y are vectors of polynomial coefficients, convolving them is equivalent to multiplying the two polynomials. An example of convolving two vectors x and y of size four is shown in Figure 1.

We exploit convolution operation to compute sliding dot products between a query (Q) and subsequences of a time series T. To achieve that, we reverse the query and pad the query with zeros to match the length of the time series T. Consider a small time series T of length four and a query Q of length two. The convolution operation between T and reversed and padded Q produces the three sliding dot products of Q over T. In addition, a few useless values are also produced, including some trailing zeroes. Thus, one convolution operation provides all sliding dot products of Q in T.

To compute the convolution operation in $O(n \log n)$, we utilize the convolution theorem [31], which states the Fourier transform of a convolution between x and y equals the pointwise multiplication of their Fourier transform. To avoid the time

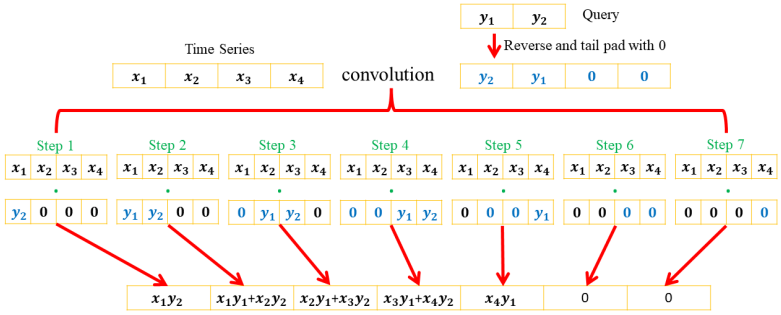

Fig. 1 Convolving a time series with a reversed and padded query produces necessary sliding dot products (results from step 2, 3, 4) for distance profile.

Algorithm 3: MASS_V1(T,Q)

Input: A time series T of length n and a query Q of length m
Output: D, the distance profile of Q in T

1 $D[1:n-m+1] \leftarrow 0$
2 $Q \leftarrow$ `zNorm(Q)`
3 $Q \leftarrow$ `reverse(Q)`
4 $Q[m+1:2*n] \leftarrow 0$ `//Tail padding zeroes`
5 $\sigma_T \leftarrow$ `movstd`(T, m)
6 $T[n+1:2*n] \leftarrow 0$ `//Tail padding zeroes`
7 $dotPs \leftarrow$ `frequencyConv`(T, Q, n, m)
8 $D \leftarrow$ `normEuclidean`$(dotPs, \sigma_T)$
9 **Function** `frequencyConv`(T, Q, n, m)
10 $TF \leftarrow$ `FFT`(T)
11 $QF \leftarrow$ `FFT`(Q)
12 $DP = TF. * QF$ `//element-wise products`
13 **return** $IFFT(DP)[m:n]$
14 **end**
15 **Function** `normEuclidean`$(dotPs, \sigma)$
`/*element-wise operations */`
16 **return** $\sqrt{2*(m-dotPs./\sigma)}$
17 **end**

domain aliasing [32] from multiplying DFTs and maintain the integrity of the convolution results, we need to zero-pad [33] x and y. This process can be summarized in Equation 5 and the detailed padding operations are described in line 4 and 6 in MASS V1 (Algorithm 3).

$$\texttt{conv}(x, y) = \texttt{IDFT}(\texttt{DFT}(\texttt{pad}(x)). * \texttt{DFT}(\texttt{pad}(y)) \tag{5}$$

MASS V1 exploits Fourier Transform, convolution theorem and cached cumulative sums for sliding standard deviation to compute the distance profile. The overall time

complexity is $O(n \log n)$ when using the `Fast Fourier Transform algorithm(FFT)` [34]. This can be seen easily because each of the lines in the algorithm, except the call to `frequencyConv` in line 7, is a linear operation with a worst-case time complexity of $O(n)$.

The Algorithm MASS V1 is rather a simplified description of the exact algorithm. There are corner cases that need separate handling. If a subsequence is a constant time series, the standard deviation is zero, causing divide by zero errors. To avoid such cases, MASS needs to check the standard deviations ahead of the division operation in Line 16. In some applications, the query Q comes from the time series T, resulting in trivial matches [5] that must be excluded by setting ∞ as distance values in the distance profile. For brevity, we omit these corner cases in the Algorithm 3.

4.2 MASS V2

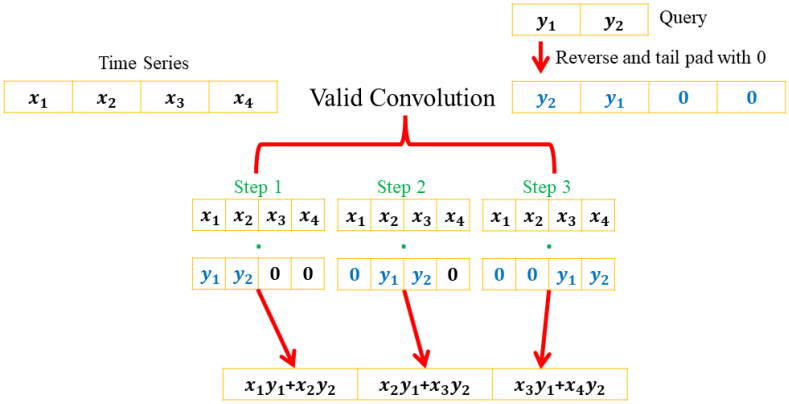

Fig. 2 Valid convolution produces necessary information for distance profiling in half space and time required by full convolution.

Algorithm 4: MASS_V2(T,Q)

Input: A time series T of length n and a query Q of length m
Output: D, the distance profile of Q in T

1 $D[1:n-m+1] \leftarrow 0$
2 $Q \leftarrow$ `zNorm(Q)`
3 $Q \leftarrow$ `reverse(Q)`
4 $Q[m+1:n] \leftarrow 0$ `//only padding query`
5 $\sigma_T \leftarrow$ `movstd(`T, m`)`
6 $dotPs \leftarrow$ `frequencyConv(`T, Q, n, m`)`
7 $D \leftarrow$ `normEuclidean(`$dotPs$, σ_T`)`

Convolution defined using DFT produces more useless numbers than the necessary ones. We can improve MASS V1 by reducing the padding size. We define an operation named valid convolution shown in Equation 6. We demonstrate MASS V2 in Algorithm 4.

$$\texttt{valid\_conv}(x, y) = \texttt{IDFT}(\texttt{DFT}(x). * \texttt{DFT}(\texttt{pad}(y))) \tag{6}$$

This operation only needs to tail pad y with zeros to match the length of x. Therefore, the output size is immediately reduced to half of that of a full convolution. This reduction does not change the overall time complexity of the algorithm; however, a $2\times$ speedup can be observed based on this simple change. Figure 2 depicts a valid convolution operation that slides the query over the time series. Note that `valid_conv(x,y)` is not symmetric and it is different from `valid_conv(y,x)`.

4.3 MASS V3

Algorithm 5: MASS_V3(T,Q)

Input: A time series T of length n, a query Q of length m and a given batch size k that $\geq$ m

Output: D, the distance profile of Q in T

$D[1 : n - m + 1] \leftarrow 0$
$Q \leftarrow$ `zNorm(Q)`
$Q \leftarrow$ `reverse(Q)`
$Q[m + 1 : k] \leftarrow 0$
$T'[1 : k] \leftarrow 0$
for $i \leftarrow 1 : k - m + 1 : n - m + 1$ **do**
    $j \leftarrow i + k - 1$
    $T'[1 : k] \leftarrow T[i : j]$
    if $j > n$ **then**
        `//handle the last batch`
        $j \leftarrow n$
        $T'[1 : k] \leftarrow 0$
        $T'[1 : j - i + 1] \leftarrow T[i : j]$
    end
    $\sigma_T \leftarrow$ `movstd(`T', m`)`
    $dotPs \leftarrow$ `frequencyConv(`T', Q, $j - i + 1$, m`)`
    $D[i:j-m+1] \leftarrow$ `normEuclidean(`$dotPs$, σ_T`)`
end

When a time series T cannot fit in the computer memory, MASS V2 will not work as defined. However, the distance profile can be computed in batches and concatenated to produce the final distance profile. In addition, it is well explored and understood that the `FFT` algorithm can benefit from aligning the input along the word boundaries in the computer memory [35]. Moreover, the latest work [36] shows that when the input size satisfies $N = 2^n$ the performance of `FFT` can be further improved by around

25%. To achieve the power of 2 input sizes and process large time series by parts, we describe the MASS V3 (Algorithm 5). In section 3.5, we show that MASS V3 has an average 31% speed improvement over MASS V2 and an average 96% improvement over our baseline just-in-time approach.

We start by describing the splitting process. We split T into segments of length that is a suitable power of two fitting in the memory. Subsequent segments must overlap $m-1$ observations to ensure we can concatenate the resulting distance profiles produced by MASS V2. The last segment can be of arbitrary length, as needed for the input time series. See Figure 3 for the splitting process.

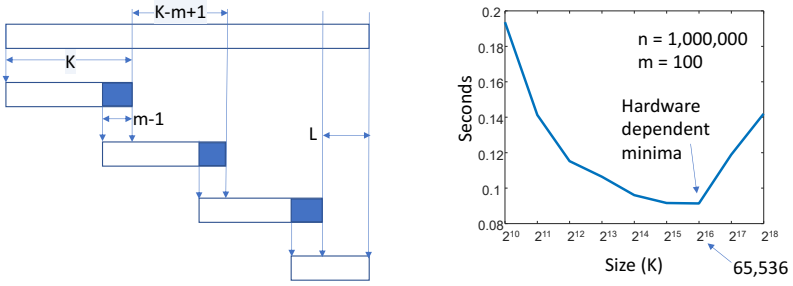

Fig. 3 (left) Splitting the long time series into segments of length K. (right) The best value of K depends on the hardware. In this example, $K = 2^{16}$ produces the fastest execution.

We explore the sensitivity of the segment size K on the overall performance of MASS V3. Figure 3 shows how the execution time of our MATLAB implementation changes when increasing K. The best batch size must vary among systems and must not have any impact on the accuracy of the output.

The linear space complexity of the algorithm ensures extreme parallelism. We can use GPUs to speed up MASS V3 by simply storing the data in GPU shared memory and using FFT operations that can exploit parallel processing on a GPU.

4.4 MASS V4

Time series similarity search is a time domain operation focusing only on the real parts of the output produced by a DFT-based fast convolution operation. Although theoretically, the output of the convolution operation must not have any imaginary part, we observe complex numbers with leakage in the imaginary part due to round-off errors. This can be eliminated by using real data `FFT` based on Hermitian symmetry. However, this solution comes with certain trade-offs. It sacrifices the simplicity of the transformation, and the space efficiency of in-place transformation [35, 37], and limits the potential for parallel computing using multiple graphics processing units [38, 39]. In addition, the complex numbers will inevitably bring in additional data structure and algorithms for related operations, which may increase the overhead cost, for instance, implementation on dedicated hardware (e.g., FPGA). We introduce a Discrete Cosine Transformation (DCT) based version of MASS that only uses the real parts of the complex numbers, and guarantees zero leakage to the imaginary parts.

Algorithm 6: MASS_V4(T,Q)

Input: A time series T of length n, a query Q of length m and a given batch size that satisfies $k \geq \lfloor (3m+1)/2 \rfloor$

Output: D, the distance profile of Q in T

$D[1:n-m+1] \leftarrow 0$
$Q \leftarrow$ `zNorm(Q)`
for $i \leftarrow 1 : k-m+1 : n-m+1$ **do**
  $j \leftarrow i+k-1$
  if $j > n$ **then**
    $j \leftarrow n$
  end
  $T' \leftarrow T[i:j]$
  $\sigma_T \leftarrow$ `movstd(`T', m`)`
  $dotPs \leftarrow$ `DCTDotProduct(`T', Q, $j-i+1$, m`)`
  $D[i{:}j-m+1] \leftarrow$ `normEuclidean(`$dotPs$, σ_T`)`
end
Function `DCTDotProduct(`T', Q, n', m`)`
  $T_{pad}, Q_{pad}, si \leftarrow$ `DCTPadding(T', Q, n', m)`
  $N \leftarrow$ `length(`T_{pad}`)`
  $T_c \leftarrow$ `DCT_type2(`T_{pad}`)` `//Orthogonal DCT applied here`
  $Q_c \leftarrow$ `DCT_type2(`Q_{pad}`)`
  $dotPs[1:N+1] \leftarrow 0$
  $dotPs[1:N] \leftarrow T_c \; .* \; Q_c$
  $dotPs[1] \leftarrow dotPs[1] * \sqrt{2}$
  $dotPs \leftarrow$ `DCT_type1(`$dotPs$`)`
  $dotPs[1] \leftarrow dotPs[1] * 2$
  return $\sqrt{2N} * dotPs[si : si+n'-m]$
end
Function `DCTPadding(`T', Q, n', m`)`
  $p_2 \leftarrow \lfloor (n'-m+1)/2 \rfloor$
  $p_1 \leftarrow p_2 + \lfloor (m+1)/2 \rfloor$
  $p_3 \leftarrow 0$
  $p_4 \leftarrow n'-m+p_1-p_2$
  $T_{pad}[1:n'+p_1] \leftarrow 0$ `//padding p1 zeros at head of T'`
  $T_{pad}[1+p_1 : n'+p_1] \leftarrow T'$
  `/*padding p2 zeros at head of Q and p4 zeros at tail */`
  $Q_{pad}[1:m+p_2+p_4] \leftarrow 0$
  $Q_{pad}[1+p_2 : 1+p_2+m-1] \leftarrow Q$
  $start\_index \leftarrow p_1 - p_2 + 1$
  return $T_{pad}, Q_{pad}, start\_index$
end

The convolution between x and y can be computed with DCT type-1 of element-wise multiplication of DCT type-2 transformed x and y with zero-padding. The

process is shown in Equation 7. The detailed padding procedures described in function `DCTPadding` from Algorithm 6.

$$\texttt{conv}(x, y) = \texttt{DCT}_1(\texttt{DCT}_2(\texttt{pad}(x)) . * \texttt{DCT}_2(\texttt{pad}(y)) \tag{7}$$

Unlike in MASS V2, where the `valid_conv` does not pad the x, MASS V4 pads both x and y. Hence, the batch size, K, to slice the time series T, must be selected considering the padded data for computing the DCT. We omit this calculation in Algorithm 6 for simplicity. However, we provide well-documented code for MASS V4 that automatically selects the best K on a given system. One worth mentioning note is that the DCT transformations in function `DCTDotProduct` are orthogonal transforms. One can think of using non-orthogonal transforms with scaling factors, however, we leave it as future work at this point.

4.5 Algorithmic Complexity

The performances of the MASS algorithms have never been documented before. In this section, we provide computational complexity, theoretical FLOPs (Floating Point Operations) count and stopwatch timing of MASS running on various data sizes.

Table 1 Time complexity in Big-O notation

Algo.	JIT	V1	V2	V3	V4
Complexity	$O\,(nm\,)$	$O\,(n \log n\,)$	$O\,(n \log n\,)$	$O\,(\frac{n-k}{k-m} k \log k\,)$	$O\,(\frac{n-k'}{k'-m} k \log k\,)$

Table 1 shows the time complexity in Big-O notation. Our baseline approach JustInTime takes $O(nm)$. V1 and V2 decrease the m term to $\log n$ since for most real-world applications $m > \log n$. V1 has a larger constant coefficient than V2 due to extra padding. For V3, the number of loops is $\lceil \frac{n-k}{k-m+1} + 1 \rceil$ k is batch size and for each loop, FFT operations take $O(k \log k)$ of length k input. V4 shares the same time complexity as V3, although the batch size k' for V4 is generally smaller than the k for V3.

Table 2 FLOPs comparison when $m = 100$, $k = 2^{15}$ for V3

n	2^{20}	2^{22}	2^{24}	2^{26}	2^{28}	2^{30}	2^{32}
JIT	2.233E8	8.934E8	3.574E9	1.429E10	5.718E10	2.287E11	9.148E11
V1	2.508E8	1.098E9	4.774E9	2.062E10	8.855E10	3.786E11	1.612E12
V2	1.263E8	5.527E8	2.401E9	1.036E10	4.450E10	1.902E11	1.612E12
V3	7.320E7	2.841E8	1.130E9	4.515E9	1.805E10	7.220E10	2.888E11
V4	1.072E8	4.200E8	1.677E9	6.704E9	2.681E10	1.072E11	4.289E11

Table 2 shows FLOPs for various time series lengths when query length m and batch size k are fixed. For the JustInTime algorithm, we count the total number of additions and multiplications; both are counted as 1 FLOP. For DFT-based MASS,

We use the state-of-the-art FLOPs formula for computing the real data FFT given by [36]. For DCT-based MASS, we applied the FLOPs formula in work [40], which is the follow-up work of [36] to compute the FLOPs count for DCT. In Table 2, we see V1 has around 12% more operation counts than JIT when $n = 2^{20}$ and 76% when $n = 2^{32}$. However, in Figure 4, we see V1 is 88% faster than JIT since additional overheads are not counted as FLOPs count, including memory operations, execution of branches and implementation of underlying libries[41]. The V3 executes the smallest FLOPs counts among all the algorithms. V4 executes 46% to 48% more FLOPs than V3 due to the extra padding with zeros and most importantly, the DCT always executes more FLOPs than FFT [36][40]. The performance show in Figure 4, matches the observations made in Table 2. Overall, V3 is the fastest in all three metrics: time complexity, stopwatch time and FLOPs count.

The code used for this experiment is available on this project site [1].

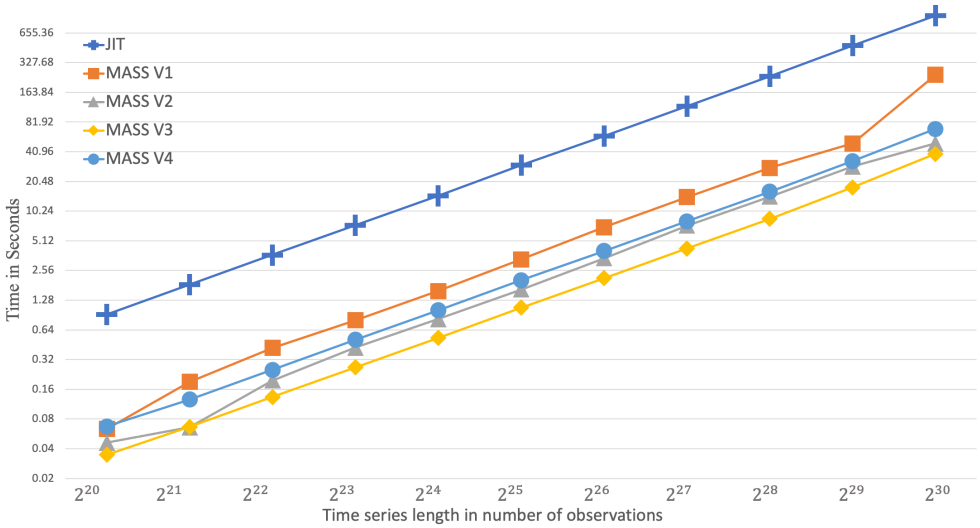

Fig. 4 The stopwatch time in seconds for different algorithms and different input lengths when $m = 100$, $k = 2^{15}$. The platform has an I9-9900k CPU with 128GB RAM, the Matlab version is 2021b.

5 Extensions of Distance Profile

5.1 Weighted Distance Profile

Often practitioners have prior knowledge about the importance of various segments in the query. To exploit that knowledge, we consider setting a weight vector $w = w[1], w[2], \ldots, w[m]$ that modifies the distance function by weighing each squared error differently, as shown in Equation 8. We can then expand the distance function in the

sum of product form, as shown in Equation 9. An application demonstrating the usage of the weighted distance profile is available in Section 7.2.

$$\texttt{weight\_dist}(x, y) = \sqrt{\sum_{i=1}^{m} w[i](\frac{x[i] - \mu_x}{\sigma_x} - y[i])^2} \tag{8}$$

$$\frac{1}{\sigma_x^2} \sum_{i=1}^{m} w[i]x[i]^2 - 2\mu_x w[i]x[i] - 2\sigma_x w[i]x[i]y[i] + \mu_x^2 w[i] + 2\mu_x \sigma_x w[i]y[i] + \sigma_x^2 w[i]y[i]^2 \tag{9}$$

In the above summation, the left three terms have $x[i]$'s, hence they need to calculate sliding dot products. The remaining terms on the right are free of x_i, hence they are pre-calculated before convolving. The first two terms are calculated by taking the sliding dot products of the weight vector over the time series x and its squared form x^2. The term $w[i]x[i]y[i]$ is calculated by taking the element-wise dot product $w \circ y$ first and then calculating the sliding dot product over the x. Note that vectors w and y are of identical size, hence, we can perform element-wise dot product.

5.2 Absolute Distance Profile

A common variation of the distance profile looks for the absolute distance between the query and the subsequences. This approach is particularly useful in two scenarios: 1) Instances where the absolute magnitudes of data points are crucial often do not require normalization. For example in aviation, particularly for flight trajectory altitude time series data, the absolute altitude values during crucial phases like takeoff and landing are more important than the normalized shape. Altitude variations must be analyzed as such to distinguish different geographical locations. Similarly, for solar power prediction involving Global Horizontal Irradiation (GHI) time series analysis also benefits from this, as absolute irradiation values directly impact power generation predictions [18]. 2) When data is pre-normalized using methods other than z-normalization, direct computation of the Euclidean distance is often adequate. An example is in cover song identification tasks [11, 12], where MASS computes the absolute distance profile on chroma-based features. These features undergo normalization using Chroma Energy Normalized Statistics (CENS), eliminating the need for additional z-normalization.

Calculating the distance profile without normalization is simpler. Only the moving sum of squares is needed in addition to the sliding dot product over the x. The distance values can be calculated using Equation. 10.

$$\texttt{absolute\_dist}(x, y) = \sqrt{\sum_{i=1}^{m} (x[i]^2 - 2x[i]y[i] + y[i]^2)} \tag{10}$$

5.3 Correlation Profile

The relationship between z-normalized Euclidean distance and Pearson's correlation enables simple transformations from one another [42].

$$\texttt{corr}(x, y) = 1 - \frac{dist(x, y)^2}{2m} \tag{11}$$

Therefore, if we have a distance profile D for a query Q and a time series T, we can produce a correlation profile in one scan. An application demonstrating the usage of the correlation profile is available in Section 7.4.

5.4 Partial Correlation Profile

Distance profiles are useful in computing the partial correlation between two time series conditioned on *any* subsequence of a longer time series. The partial correlation coefficient between two variables x and y, conditioned on a variable z is defined in Equation 12.

$$\rho_{xy.z} = \frac{\rho_{xy} - \rho_{xz}\rho_{yz}}{\sqrt{1 - \rho_{xz}^2}\sqrt{1 - \rho_{yz}^2}} \tag{12}$$

Correlation profiles can be extended to compute the partial correlation profile between two queries with respect to a long conditional time series z. The computation can be done by computing the correlation profiles of x and y as queries in the time series z. This provides ρ_{xz} and ρ_{yz} in the above equation. Since ρ_{xy} is constant for queries x and y, the partial correlation profile can, thus, be calculated from the two correlation profiles.

Partial correlation is crucial in scenarios where it's necessary to measure the correlation between two variables while controlling the effects of a third variable. This measurement is especially relevant in cases where external factors influence the correlation between primary variables. For instance, in the analysis of ECG patterns, considering additional factors such as air pressure or body posture may be needed. Omitting such factors can lead to inaccurate correlations. To illustrate the significance of the partial correlation profile, we employed a 1-NN classifier, comparing its accuracy using both partial and standard correlation profiles. The methodology is detailed in Algorithm 7. We tested on the "ECG200" dataset from the UCR repository [43], which includes data classified into two categories: 'normal' and 'ischemia', with samples from both male and female subjects. We initially computed the standard correlation profile between the testing and training datasets, employing labels from the nearest neighbors for classification. This standard approach yielded an accuracy of 88%. However, when incorporating gender as a factor through partial correlation, the accuracy improved. For each test case, we computed the partial correlation with respect to all training set cases to accommodate the absence of explicit gender information in the dataset. The minimum correlation value was then selected, as specified in line 15 of Algorithm 7. This incorporation of gender via partial correlation led to a 3% increase in accuracy compared to the standard correlation approach.

Algorithm 7: Partial_Corr_Classification(X_train, X_test, Y_train, Y_test)

Input: X_train is the training data with size $N\_train \times m$.
X_test is the testing data with size $N\_test \times m$.
m is the time series length of each case.
Output: Accuracy of the 1-NN classifier with the partial correlation profile.

$Train\_TS \leftarrow$ `concat(`X_train`)` `//concatenating rows in X_train`
$mask \leftarrow [1 : m : N\_train * m - m + 1]$
$M[1 : N\_test, 1 : N\_train] \leftarrow 0$
for $i \leftarrow 1 : N\_Test$ **do**
  $x\_test \leftarrow X\_test[i, :]$
  $C_1 \leftarrow$ `MASSCorr(`$Train\_TS$, x_test`)`
  $C_1 \leftarrow C_1[mask]$
  for $j \leftarrow 1 : N\_Train$ **do**
    $x\_train \leftarrow X\_train[j, :]$
    $c_1 \leftarrow C_1[j]$
    $C_2 \leftarrow$ `MASSCorr(`$Train\_TS$, x_train`)`
    $C_2 \leftarrow C_2[mask]$
    `//`$.*$ `is the Hadamard production`
    $partical\_corr \leftarrow (c_1 - C_1 .* C_2)./\sqrt{(1 - C_1^2) .* (1 - C_2^2)}$
    $partical\_corr[j] \leftarrow Inf$
    $M[i, j] \leftarrow$ `min(`$partical\_corr$`)`
  end
end
$LOC \leftarrow$ `argmax(`M`)` `//get index of maximum element in each row.`
$\hat{Y} \leftarrow Y\_train[LOC]$
$a \leftarrow Y\_test - \hat{Y}$
$acc \leftarrow$ `sum(`$a \neq 0$`)`
return acc
Function `MASSCorr(`x, y`)`
  $D \leftarrow$ `MASS(`x, y`)`
  $C \leftarrow 1 - D^2./(2*$`length(`$y$`))` `//element-wise operations, Equation 11`
  return C
end

5.5 Multivariate Time Series

In certain applications, the analysis of multivariate time series data becomes essential. One common approach for computing the distance profile of multivariate time series is first computing the individual profiles of each dimension, and then integrating them into a single, unified profile. The integration process typically involves merging these profiles using a weighting factor [14, 44]. The weighting factor can either be assigned uniformly across all dimensions or determined through a learning process from the data.

5.6 Discussion

All the versions of distance profiles can be efficiently computed using any version of the MASS algorithm with little modifications, which are highlighted in Equations 9, 10, 11, 11. A key advantage of using MASS for these calculations is its consistency in both space and time complexity across different types of correlation profiles. The exact time complexity is available in Table 1.

6 Comparison with Indexing Solution

Distance profiling is different from indexing or index-based solutions to search for the nearest neighbor. This is an important distinction to be made, hence, demands a separate section.

An index is built on a large amount of data (typically larger than memory) in order to search the nearest neighbor (or k-nearest neighbors) of any given query very efficiently [45][46]. There are several parameters involved in building and using an index: the time to build the index, the time to search for one query, and the number and types of queries searched. The goal is to search for a query interactively, while the time to create an index is generally long because of the involvement of the disk. Most indexing works consider a wide variety of queries to demonstrate generalizability.

In practice, if an index is already built and only a few neighbors are needed for each query, distance profiling is not suitable. In contrast, if the data and the query both change frequently, distance profiling is suitable to offset the overhead of index creation. What is the largest sized data that we can profile at interactive speed? Roughly, profiling a one billion long time series takes less than a minute on an off-the-shelf computer. We argue that any *time series subsequence* database less than one billion observations does not warrant indexing.

7 Utility in Application Domains

The utility of a distance profile comes from the knowledge of the entire distribution of distances between a query and a time series. When the frequency of the matches is important and variable for different queries, the distance profile is a great tool to exploit. We show three use cases of distance profiles in three different domains: robotics, seismology and power grid management.

7.1 Survey on applications

7.2 Robots

The accelerometer on a foot of a Sony AIBO robot records the walking cycles of the robot. The accelerometer readings show signatures produced by the surface via the reactive force on the foot. We take two cycles of a robot walking on a carpet and use a weighted distance profile by having zero weightage for the segment when the foot is in the air. The distance profile shows matches when the robot was walking on a carpet.

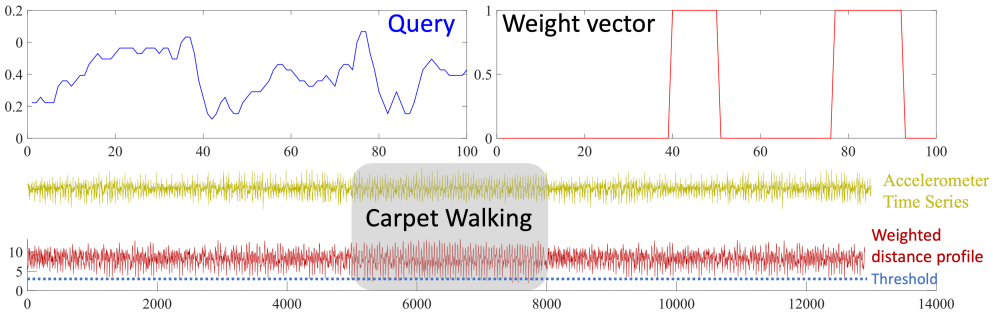

Fig. 5 The query corresponds to the accelerometer signal of two steps taken by a SONY robot walking on the carpet. The accompanying weighted vector preserves the periods where the robot's foot contacts the carpet and ignores periods where the foot is in the air. We applied MASS to calculate the weighted distance profile of the query on the full sequence of the robot's movements over time (yellow time series), and the red time series is the result distance profile. In this profile, lower values correspond to a closer match with the query signal. We use a grey-shaded area to denote the time when the robot is moving on the carpet based on the ground truth data. The weighted distance profile matches this ground truth, as multiple subsequences in this period have a weighted distance below the threshold.

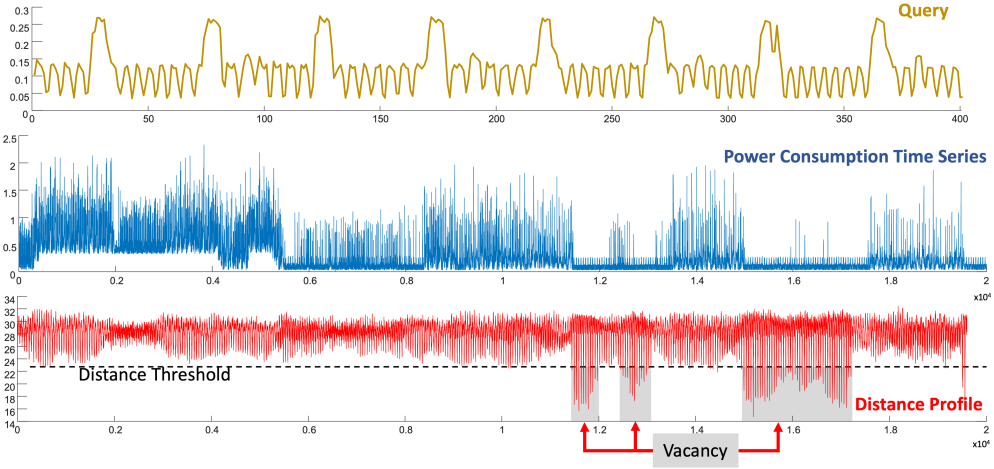

Fig. 6 The top plot illustrates the query pattern which represents the typical power consumption of a house during idle or vacant periods. This pattern is characterized by its cyclic and consistent nature with minimal variations, which is typical of a household when it is not actively being used. The middle plot exhibits the power consumption time series data of a household over an extended period. The fluctuations and spikes indicate varying levels of activity and usage within the household. In the bottom plot, the distance profile calculated by the MASS algorithm is presented. The dashed line represents a distance threshold set at 22.5. Periods, where the distance profile falls below this threshold, are shaded in gray and labeled as Vacancy. These intervals signify times when the household's power consumption pattern closely matches the idle or vacant query pattern, suggesting that the house was likely unoccupied during these times.

7.3 Solar Power

MASS can be used to create a distance profile of a power consumption time series [47]. The total power consumption of a household contains patterns of the idle state of the house (i.e. when nobody is in the house). If the idle state pattern of a house is known, the distance profile of the power consumption time series easily provides the number of days (or percentage of days) the household was vacant. In figure 6, we show a power consumption time series for six months and an idle state pattern. We also show the distance profile, which can be thresholded at 22.8 to calculate the proportion of time the house was vacant. In this example, 18.2% of the time, the house was vacant.

7.4 Seismology

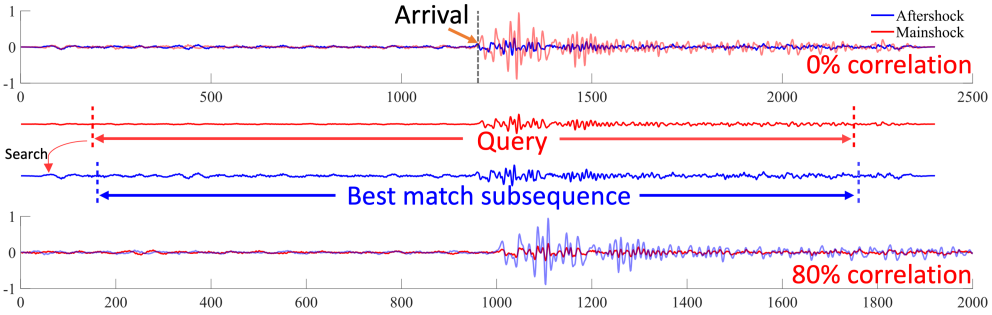

Fig. 7 Two seismograph traces are presented at the top, the red trace denotes the mainshock, while the blue trace represents an aftershock. These traces are extracted from a streaming time series recorded at a seismic station running at 40 Hz. The displayed data includes the 30 seconds before and after the arrival time of the seismic event, which has been identified by human analysts. Initially, when compared directly, the two sequences exhibit a 0% correlation, indicating no direct similarity between the mainshock and aftershock as captured by the seismographs. However, we use MASS to compute the distance profile of a query, which is a 40-second subsequence from the mainshock, against the 60-second sequence of the aftershock. The bottom figure showcases the 80% correlation between the query and the best-matching subsequence from the aftershock data.

MASS can be used to match aftershocks to each other when a large earthquake happens. Consider the two seismographs in Figure 7 recorded at the station MKAR in Kazakhstan. The aftershock (shown in blue) does not align properly with the red major earthquake (mainshock) if we use human-annotated wave arrival time [48]. In this case, the arrival times are more than 250ms apart, resulting in a poor correlation (0.08) coefficient. When we use sliding (or shifting) Euclidean distance [49], the two aftershocks show an 80% correlation that confirms the closeness of the sources. MASS is a great tool to compute sliding Euclidean distance in $O(n \log n)$ time as opposed to the naive nested-loop algorithm.

8 Utility in Algorithms

Distance profiles are good data structures to redesign and speed up existing data mining algorithms. We pick Time Series Discord [50] and clustering [51] to demonstrate the utility.

Algorithm 8: MASS_Discord(T, m, $best\_so\_far$)

Input: A time series T of length n and a discord length m
Output: *discordLocation* and *discordDistance* in T

$q \leftarrow T[1:m]$
$i \leftarrow 1$
$iLoc \leftarrow [1, 2, \ldots, n-m+1]$
$discordDistance \leftarrow best\_so\_far$
$discordLoc \leftarrow 0$
while $i <= n-m+1$ **do**
  $D \leftarrow MASS(T, q)$
  $D[max(i-m+1, 1) : min(i+m-1, n)] \leftarrow \infty$
  if $minimum(D) > discordDistance$ **then**
    $discordDistance \leftarrow minimum(D)$
    $discordLoc \leftarrow i$
  end
  for $j \leftarrow 1 : n-m+1$ **do**
    if $D[j] < discordDistance$ **then**
      $iLoc[j] \leftarrow -1$
    end
  end
  $i \leftarrow$ `Next positive index in` $iLoc$
  $q \leftarrow T[i : i+m-1]$
end

Time series discord is the subsequence in a time series that has the furthest nearest neighbor. The subsequence, whose nearest neighbor is the most dissimilar, is the time series discord. Traditional discord discovery algorithms exploit pruning and early abandoning strategies when computing individual distances. However, distance profiles can merge many of these distance computations and speed up the search for discord.

In Algorithm 8, we show how distance profiles can be used to prune unpromising subsequences and hop over regions of repetitive segments of the time series. The efficiency of the algorithm depends on data characteristics in the same way traditional algorithms depend. However, a *traditional* algorithm computes one distance at a time and occasionally prunes candidate subsequences. The MASS_Discord computes one distance profile at a time and prunes as many candidates as possible. Thus, MASS_Discord spends less time in distance computation and more time in pruning. We see a sizable speed-up over traditional pruning strategies when tested on the three real datasets described in the previous section. In Figure 8, we vary the length of the

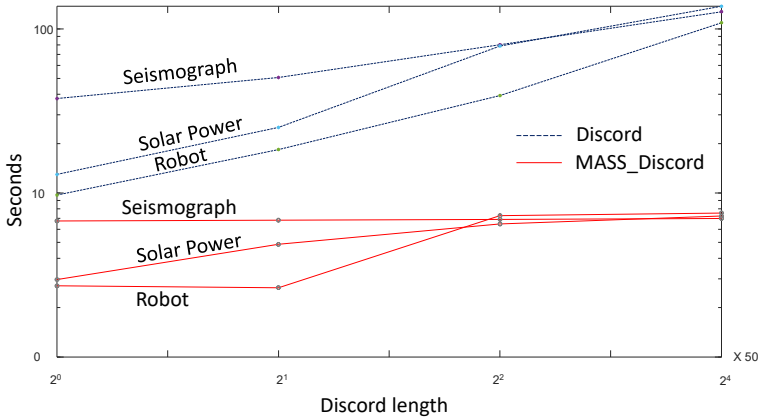

Fig. 8 Execution time of MASS_Discord in comparison to traditional discord discovery algorithm on three real datasets.

discord and measure the time in seconds to find the discord by both traditional and MASS_Discord algorithms.

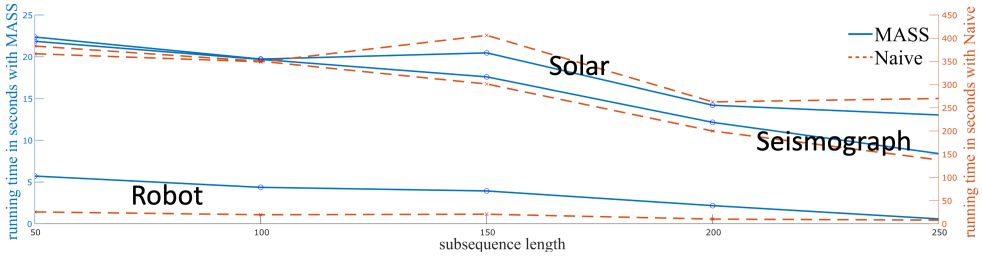

Fig. 9 Execution time comparison between MASS and a naive approach when applied to complete-linkage clustering on streaming time series. The comparison spans three different real-world datasets robots, solar data, and seismographs. The MASS algorithm demonstrates a considerable reduction in execution time compared to the naive method, maintaining a consistent lead as the subsequence length increases. The MASS-based approach is around 20 times faster on solar datasets and approximately 15 times faster on robot and seismograph datasets.

The task of clustering within a single time series stream involves grouping subsequences from the stream in a manner where the selected subsequences are non-overlapping and may have gaps between them. This approach, based on the principle that clustering of time series from a single stream of data requires ignoring some of the data [51].

Contrasts with the clustering of individual time series, which requires computing distances among independent time series of the same length. For time series stream clustering, it is essential to consider the distance between all possible pairs of subsequences. One method is to maintain all pairwise distances in a matrix, but this approach has significant space complexity, $O(n^2)$ where n is the length of the time series. For instance, a time series with a length of 46,340 would require approximately

Algorithm 9: MASS_Complete-linkage_Clustering(T, m, δ)

Input: A time series T of length n, subsequence length m and a threshold of maximum distance in a cluster.
Output: C

```
//First compute matrix profile.
MP, MPI ← SCRIMP++(T, m)
//iLoc maintains valid index in MP.
iLoc ← ones(n − m + 1)
for i ← 1 : n − m + 1:
    C[i] ← {};
while sum(iLoc) > 1 and min(MP) < δ:
    cmi ← argmin(MP)
    n1 ← min(cmi, MPI[cmi])
    n2 ← max(cmi, MPI[cmi])
    nc ← n1
    merge(C, m, n1, n2, iLoc)
    md, mdi ← Update(T, m, C, n1, iLoc)
    MP[n1] ← md
    MPI[n1] ← mdi
    //Update elements in MP, MPI
     whose NN location is not valid.
    for k ← 1 : n − m + 1:
        if iLoc[k] ≠ 0 and
         (iLoc[MPI[k]] == 0 or
         MPI[k] == nc):
            MP[k], MPI[k] ← Update(T, m,
             C, n1, iLoc)
    MP[iLoc == 0] ← inf
    MPI[iLoc == 0] ← inf
return C

Function merge(C, m, n1, n2, iLoc)
    if empty(C[n1]) and empty(C[n2]):
        C[n1] ← {n1, n2}
        iLoc[n1−m+1 : n1+m+1] ← 0
        iLoc[n2−m+1 : n2+m+1] ← 0
        iLoc[n1] ← 1
    elif empty(C[n1]) and !empty(C[n2]):
        C[n1] ← C[n2] ∪ {n1}
        C[n2] ← {}
        iLoc[n1 − m + 1 : n1 + m + 1] ← 0
        iLoc[n1] ← 1
        iLoc[n2] ← 0
    elif !empty(C[n1]) and empty(C[n2]):
        C[n1] ← C[n1] ∪ {n2}
        iLoc[n2 − m + 1 : n2 + m + 1] ← 0
    else:
        C[n1] ← C[n1] ∪ C[n2]
        C[n2] ← {}
        iLoc[n2] ← 0
Function Update(T, m, C, ti, iLoc)
    //ti is the target index in MP
     pending to be updated.
    if empty(C[ti]):
        //ti is a sub-sequence
        D ← MASS(T, T[ti:ti+m−1])
        D[ti−m+1 : ti+m−1] ← Inf
    else:
        //ti is a cluster
        D ← −Inf(n−m+1)
        for k ← 1: length(C[ti]):
            si ← C[ti][k]
            cur_dist ← MASS(T, T(si:
             si+m−1))
             //element-wise maximum
            D ← max(D, cur_dist)
        D[ti] ← Inf
    for i ← 1 : length(D):
        if !empty(C[i]) and i ≠ ti:
            for k ← 1 : length(C[i]):
                D[i] ← max([D[i]], D[C[i][k]])
    D[iLoc == 0] ← Inf
    [dm, dmi] ← min(D)
    return dm,dmi
```

16GB of RAM for the distance matrix. Thus, an algorithm with $O(n)$ space complexity is more practical for real-world applications.

We propose a complete-linkage clustering method for streaming time series that utilizes only $O(n)$ space, leveraging the MASS in Algorithm 9. The process starts by

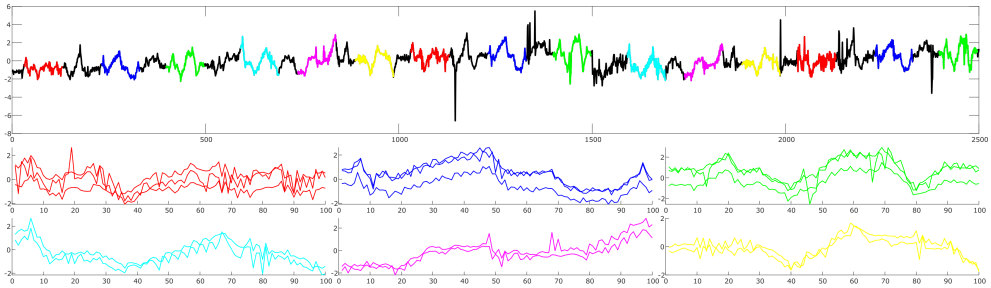

Fig. 10 The top graph shows the time series of the first dimension of the Winding dataset with various segments color-coded to represent different clusters identified by Algorithm 9. The lower graphs provide expanded views of identified clusters, showing the consistency within each cluster despite the noisy environment. The subsequence length is set to 100.

computing the matrix profile using the SCRIMP++ algorithm [9] (line 1). In each iteration of the While loop (line 5), a new cluster is formed or two existing clusters are merged based on the matrix profile, as defined in the merge function (starting at line 20). Any changes in the clusters trigger the update function (starting at line 39), which updates certain matrix profile cells. Here, we use MASS to compute the distance profile, and then employ a MAX aggregator to update cluster distances. This process requires only n auxiliary space since only one distance profile is maintained at a time. Overall, the space requirement to store $iLoc, MP, MPI, D$ is $4n$.

We benchmarked the performance of our MASS based algorithm against a naive implementation that uses the `pdist` function in MATLAB to only compute the necessary pairwise distances. This comparison was conducted by varying the length of the subsequence and measuring the processing time in seconds for both algorithms on three real-world datasets. Figure 9 demonstrates that MASS based approach is 20 times faster on solar dataset and around 15 times faster on robot and seismograph datasets.

In Figure 10 we show the results of clustering an industrial dataset. The data comes from an industrial wire winding process [52]. Note that the data has significant non-uniform noise, including spikes and dropouts. Although ground truth data is not available for verification, the clustering results visually demonstrate a clear distinction between the different clusters, suggesting that the applied method effectively captures the inherent similarities within clusters and differences across clusters.

9 Conclusion

We define the distance profile of a query over a time series and provide a series of algorithms to compute distance profiles under Euclidean distance and its variants. We discuss the performance of these algorithms both quantitatively and qualitatively. We demonstrate the utility of distance profiles as a tool for data mining algorithms in various real applications. The paper serves as the first complete documentation of distance profiling algorithms, which had only partially been discussed in articles and web pages.

Acknowledgments. This material is based on work supported by the National Science Foundation under #2104537.

References

[1] Mueen, A., Zhu, Y., Yeh, M., Kamgar, K., Viswanathan, K., Gupta, C., Keogh, E.: The Fastest Similarity Search Algorithm for Time Series Subsequences under Euclidean Distance. http://www.cs.unm.edu/~mueen/FastestSimilaritySearch.html (2017)

[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. Data Mining and Knowledge Discovery (2017) https://doi.org/10.1007/s10618-017-0519-9

[3] Zhu, L., Lu, C., Sun, Y.: Time Series Shapelet Classification Based Online Short-Term Voltage Stability Assessment. IEEE Transactions on Power Systems **31**(2), 1430–1439 (2016) https://doi.org/10.1109/TPWRS.2015.2413895

[4] Zhu, Y., Mueen, A., Keogh, E.: Admissible time series motif discovery with missing data (2018)

[5] Yeh, C.-C.M., Zhu, Y., Ulanova, L., Begum, N., Ding, Y., Dau, H.A., Silva, D.F., Mueen, A., Keogh, E.: Matrix profile I: All pairs similarity joins for time series: A unifying view that includes motifs, discords and shapelets. In: Proceedings - IEEE International Conference on Data Mining, ICDM (2017). https://doi.org/10.1109/ICDM.2016.89

[6] Keogh, E.: The UCR Matrix Profile Page. https://www.cs.ucr.edu/~eamonn/MatrixProfile.html (2017)

[7] Wilhelm, S., Kasbauer, J.: Exploiting smart meter power consumption measurements for human activity recognition (har) with a motif-detection-based non-intrusive load monitoring (nilm) approach. Sensors **21**(23) (2021) https://doi.org/10.3390/s21238036

[8] Chandrasekar, S., Coble, J.B., List, F., Carver, K., Beauchamp, S., Godfrey, A., Paquit, V., Babu, S.S.: Similarity analysis for thermal signature comparison in metal additive manufacturing. Materials & Design **224**, 111261 (2022) https://doi.org/10.1016/j.matdes.2022.111261

[9] Zhu, Y., Yeh, C.-C.M., Zimmerman, Z., Kamgar, K., Keogh, E.: Matrix profile xi: Scrimp++: Time series motif discovery at interactive speeds. In: 2018 IEEE International Conference on Data Mining (ICDM), pp. 837–846 (2018). https://doi.org/10.1109/ICDM.2018.00099

[10] Zhu, Y., Zimmerman, Z., Senobari, N.S., Yeh, C.-C.M., Funning, G., Mueen, A.,

Brisk, P., Keogh, E.: Matrix profile ii: Exploiting a novel algorithm and gpus to break the one hundred million barrier for time series motifs and joins. In: 2016 IEEE 16th International Conference on Data Mining (ICDM), pp. 739–748 (2016). https://doi.org/10.1109/ICDM.2016.0085

[11] Silva, D.F., Yeh, C.M., Batista, G.E.A.P.A., Keogh, E.J.: Simple: Assessing music similarity using subsequences joins. In: Mandel, M.I., Devaney, J., Turnbull, D., Tzanetakis, G. (eds.) Proceedings of the 17th International Society for Music Information Retrieval Conference, ISMIR 2016, New York City, United States, August 7-11, 2016, pp. 23–29 (2016). https://wp.nyu.edu/ismir2016/wp-content/uploads/sites/2294/2016/07/099_Paper.pdf

[12] Silva, D.F., Yeh, C.-C.M., Zhu, Y., Batista, G.E.A.P.A., Keogh, E.: Fast similarity matrix profile for music analysis and exploration. IEEE Transactions on Multimedia **21**(1), 29–38 (2019) https://doi.org/10.1109/TMM.2018.2849563

[13] Uudeberg, T., Belikov, J., Päeske, L., Hinrikus, H., Liiv, I., Bachmann, M.: In-phase matrix profile: A novel method for the detection of major depressive disorder. Biomedical Signal Processing and Control, 105378 (2023) https://doi.org/10.1016/j.bspc.2023.105378

[14] Yang, D., Alessandrini, S.: An ultra-fast way of searching weather analogs for renewable energy forecasting. Solar Energy **185**, 255–261 (2019) https://doi.org/10.1016/j.solener.2019.03.068

[15] Abdoli, A., Alaee, S., Imani, S., Murillo, A., Gerry, A., Hickle, L., Keogh, E.: Fitbit for chickens? time series data mining can increase the productivity of poultry farms. In: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. KDD '20, pp. 3328–3336. Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3394486.3403385 . https://doi.org/10.1145/3394486.3403385

[16] Heo, H., Kim, H.J., Kim, W.S., Lee, K.: Cover song identification with metric learning using distance as a feature. In: Cunningham, S.J., Duan, Z., Hu, X., Turnbull, D. (eds.) Proceedings of the 18th International Society for Music Information Retrieval Conference, ISMIR 2017, Suzhou, China, October 23-27, 2017, pp. 628–634 (2017). https://ismir2017.smcnus.org/wp-content/uploads/2017/10/33_Paper.pdf

[17] Mercer, R., Alaee, S., Abdoli, A., Senobari, N.S., Singh, S., Murillo, A., Keogh, E.: Introducing the contrast profile: a novel time series primitive that allows real world classification. Data Mining and Knowledge Discovery **36**, 877–915 (2022) https://doi.org/10.1007/s10618-022-00824-5

[18] Yang, D., Wu, E., Kleissl, J.: Operational solar forecasting for the real-time market. International Journal of Forecasting **35**(4), 1499–1519 (2019) https://doi.org/10.1016/j.ijforecast.2019.03.009

[19] Yang, D.: Ultra-fast preselection in lasso-type spatio-temporal solar forecasting problems. Solar Energy **176**, 788–796 (2018) https://doi.org/10.1016/j.solener.2018.08.041

[20] Franch, G., Jurman, G., Coviello, L., Pendesini, M., Furlanello, C.: Mass-umap: Fast and accurate analog ensemble search in weather radar archives. Remote Sensing **11**(24) (2019) https://doi.org/10.3390/rs11242922

[21] Mercer, R., Keogh, E.: Matrix profile xxv: Introducing novelets: A primitive that allows online detection of emerging behaviors in time series. In: 2022 IEEE International Conference on Data Mining (ICDM), pp. 338–347 (2022). https://doi.org/10.1109/ICDM54844.2022.00044

[22] Alshaer, M., Garcia-Rodriguez, S., Gouy-Pailler, C.: Detecting anomalies from streaming time series using matrix profile and shapelets learning. In: 2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI), pp. 376–383 (2020). https://doi.org/10.1109/ICTAI50040.2020.00066

[23] Kammerer, K., Hoppenstedt, B., Pryss, R., Stökler, S., Allgaier, J., Reichert, M.: Anomaly detections for manufacturing systems based on sensor data—insights into two challenging real-world production settings. Sensors **19**(24) (2019) https://doi.org/10.3390/s19245370

[24] Lu, Y., Wu, R., Mueen, A., Zuluaga, M.A., Keogh, E.: Matrix profile xxiv: Scaling time series anomaly detection to trillions of datapoints and ultra-fast arriving data streams. In: Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. KDD '22, pp. 1173–1182. Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3534678.3539271 . https://doi.org/10.1145/3534678.3539271

[25] Keogh, E.J., Pazzani, M.J.: Scaling up dynamic time warping for datamining applications. In: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '00, pp. 285–289. ACM Press, New York, New York, USA (2000). https://doi.org/10.1145/347090.347153 . http://dl.acm.org/citation.cfm?id=347090.347153

[26] Vlachos, M., Hadjieleftheriou, M., Gunopulos, D., Keogh, E.: Indexing Multidimensional Time-series with Support for Multiple Distance Measures. In: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD '03, pp. 216–225. ACM, New York, NY, USA (2003). https://doi.org/10.1145/956750.956777 . http://doi.acm.org/10.1145/956750.956777

[27] Stefan, A., Athitsos, V., Das, G.: The Move-Split-Merge Metric for Time Series. IEEE Transactions on Knowledge and Data Engineering **25**(6), 1425–1438 (2013) https://doi.org/10.1109/TKDE.2012.88

[28] Silva, D.F., Batista, G.E.A.P.A., Keogh, E.: Prefix and Suffix Invariant Dynamic Time Warping. In: 2016 IEEE 16th International Conference on Data Mining (ICDM), pp. 1209–1214 (2016). https://doi.org/10.1109/ICDM.2016.0161

[29] Mueen, A., Keogh, E.: Online discovery and maintenance of time series motifs. In: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '10, p. 1089. ACM Press, New York, New York, USA (2010). https://doi.org/10.1145/1835804.1835941 . http://dl.acm.org/citation.cfm?id=1835804.1835941

[30] Rakthanmanon, T., Campana, B., Mueen, A., Batista, G., Westover, B., Zhu, Q., Zakaria, J., Keogh, E.: Searching and mining trillions of time series subsequences under dynamic time warping. In: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 262–270 (2012). https://doi.org/10.1145/2339530.2339576

[31] Arfken, G.B., Weber, H.J., Harris, F.E.: Chapter 20 - integral transforms. In: Arfken, G.B., Weber, H.J., Harris, F.E. (eds.) Mathematical Methods for Physicists (Seventh Edition), Seventh edition edn., pp. 963–1046. Academic Press, Boston (2013). https://doi.org/10.1016/B978-0-12-384654-9.00020-7

[32] HARRIS, F.J.: Chapter 8 - time domain signal processing with the dft. In: Elliott, D.F. (ed.) Handbook of Digital Signal Processing, pp. 633–699. Academic Press, San Diego (1987). https://doi.org/10.1016/B978-0-08-050780-4.50013-8 . https://www.sciencedirect.com/science/article/pii/B9780080507804500138

[33] Lai, E.: 4 - frequency-domain representation of discrete-time signals. In: Lai, E. (ed.) Practical Digital Signal Processing, pp. 61–78. Newnes, Oxford (2003). https://doi.org/10.1016/B978-075065798-3/50004-7 . https://www.sciencedirect.com/science/article/pii/B9780750657983500047

[34] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. The MIT Press, Cambridge (2009)

[35] Fftw: An adaptive software architecture for the fft, vol. 3 (1998). https://doi.org/10.1109/ICASSP.1998.681704

[36] Johnson, S.G., Frigo, M.: A modified split-radix fft with fewer arithmetic operations. IEEE Transactions on Signal Processing **55** (2007) https://doi.org/10.1109/TSP.2006.882087

[37] Frigo, M., Johnson, S.G.: FFTW manual. https://fftw.org/fftw3.pdf (2020)

[38] Fast Fourier Transform with CuPy. https://docs.cupy.dev/en/stable/user_guide/fft.html (2023)

[39] Multiple GPU cuFFT Transforms. https://docs.nvidia.com/cuda/cufft/index.

html#multiple-gpu-2d-and-3d-transforms-on-permuted-input (2023)

[40] Shao, X., Johnson, S.G.: Type-ii/iii dct/dst algorithms with reduced number of arithmetic operations. Signal Processing **88** (2008) https://doi.org/10.1016/j.sigpro.2008.01.004

[41] Frigo, M., Johnson, S.G.: The design and implementation of fftw3, vol. 93 (2005). https://doi.org/10.1109/JPROC.2004.840301

[42] Mueen, A., Nath, S., Liu, J.: Fast approximate correlation for massive time-series data. In: Proc. ACM SIGMOD Int. Conf. on Management of Data, pp. 171–182 (2010). https://doi.org/10.1145/1807167.1807188

[43] Anthony Bagnall, W.V. Jason Lines, Keogh, E.: The UEA & UCR Time Series Classification Repository (2023). www.timeseriesclassification.com

[44] Piatov, D., Helmer, S., Dignös, A., Gamper, J.: Interactive and space-efficient multi-dimensional time series subsequence matching. Information Systems **82**, 121–135 (2019) https://doi.org/10.1016/j.is.2018.08.002

[45] Shieh, J., Keogh, E.: iSAX : Indexing and Mining Terabyte Sized Time Series. In: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, vol. KDD '08, pp. 623–631 (2008). https://doi.org/10.1145/1401890.1401966 . http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.155.4531

[46] Camerra, A., Palpanas, T., Shieh, J., Keogh, E.: iSAX 2.0: Indexing and Mining One Billion Time Series. In: 2010 IEEE International Conference on Data Mining, pp. 58–67 (2010). https://doi.org/10.1109/ICDM.2010.124

[47] Mollah, M.P., Souza, V.M.A., Mueen, A.: Multi-way Time Series Join on Multi-length Patterns. In: 2021 IEEE International Conference on Data Mining (ICDM), pp. 429–438 (2021). https://doi.org/10.1109/ICDM51629.2021.00054

[48] Zhong, S., Souza, V.M.A., Mueen, A.: FilCorr: Filtered and Lagged Correlation on Streaming Time Series. In: 2020 IEEE International Conference on Data Mining (ICDM), pp. 1436–1441 (2020). https://doi.org/10.1109/ICDM50108.2020.00190

[49] Mueen, A., Keogh, E., Young, N.: Logical-shapelets: an expressive primitive for time series classification. the 17th ACM SIGKDD international conference, 1154–1162 (2011) https://doi.org/10.1145/2020408.2020587

[50] Yankov, D., Keogh, E., Rebbapragada, U.: Disk aware discord discovery: Finding unusual time series in terabyte sized datasets. In: Knowledge and Information Systems, vol. 17, pp. 241–262 (2008). https://doi.org/10.1007/s10115-008-0131-9

[51] Rakthanmanon, T., Keogh, E.J., Lonardi, S., Evans, S.: Time series epenthesis:

Clustering time series streams requires ignoring some data. In: Proceedings - IEEE International Conference on Data Mining, ICDM. ICDM '11, pp. 547–556 (2011). https://doi.org/10.1109/ICDM.2011.146

[52] Bastogne, T., Noura, H., Richard, A., Hittinger, J.-M.: Application of subspace methods to the identification of a winding process. In: 1997 European Control Conference (ECC), pp. 2168–2173 (1997). https://doi.org/10.23919/ECC.1997.7082426