FSMDTW: A Fast Index-free Subsequence Matching Algorithm for Dynamic Time Warping

Zemin Chao
Harbin Institute of Technology
Harbin, Heilongjiang, China
chaozm@hit.edu.cn

Qiaoyi Zheng
Harbin Institute of Technology
Harbin, Heilongjiang, China
2022111568@stu.hit.edu.cn

Zhixin Qi
Harbin Institute of Technology
Harbin, Heilongjiang, China
qizhx@hit.edu.cn

Hongzhi Wang
Harbin Institute of Technology
Harbin, Heilongjiang, China
wangzh@hit.edu.cn

ABSTRACT

The subsequence matching problem utilizing dynamic time warping as the similarity measurement has been recognized as a key operation in time series analysis for more than two decades. Existing index-free algorithms depend on DTW lower bounds to discard the unpromising candidate. However, these approaches typically cost $O(m)$ time for each candidate, where m is the length of the query. Consequently, the overhead of computing the DTW lower bounds occupies a significant portion of the time in subsequence matching tasks. This paper proposes new algorithms capable of computing the DTW lower bounds in average $O(\log m)$ time for each candidate, substantially alleviating this bottleneck of the subsequence matching problem. In addition, this paper designs novel DTW lower bounds according to the characteristics of the subsequence matching problem, which is more effective without introducing significant computational overhead. Based on the above improvements, an efficient subsequence matching algorithm called FSMDTW is designed. Experiments conducted on both real and synthetic datasets show that the proposed algorithm is about 2.6 times faster than SOTA on short and medium-length queries and up to one order of magnitude faster on longer queries.

PVLDB Reference Format:
Zemin Chao, Qiaoyi Zheng, Zhixin Qi, and Hongzhi Wang. FSMDTW: A Fast Index-free Subsequence Matching Algorithm for Dynamic Time Warping. PVLDB, 18(10): 3628 - 3640, 2025.
doi:10.14778/3748191.3748220

PVLDB Artifact Availability:
The source code, data, and / or other artifacts have been made available at https://github.com/QW1k0YA/A-Fast-Index-free-Subsequence-Matching-Algorithm-for-Dynamic-Time-Warping.

1 INTRODUCTION

Given time series $S \in \mathbb{R}^n$, a query $Q \in \mathbb{R}^m$, and a threshold $\epsilon > 0$, the subsequence matching problem is to find all subsequences of S that are similar enough to Q, as depicted in Figure 1. Dynamic Time Warping (DTW) is known as a superior similarity measurement in many real-world applications [2, 3, 25, 26]. Consequently, subsequence matching using DTW is considered one of the most important problems in time series analysis [23, 36], and it is widely used in disease diagnosis [19], automatic data annotation [10], and EEG analysis [5].

Figure 1: An illustrative example of the subsequence matching problem, which is to find subsequences (continuous segments) in time series S that are similar to query Q.

Existing subsequence matching algorithms for DTW can be categorized into index-based or index-free algorithms. However, all known index-based algorithms are constrained by either the length of the query or the maximum extent of uniform scaling [6, 17, 32, 36], and they also require additional space and time for indexing time series. Furthermore, time series can only be accessed once in stream processing, making index-based algorithms unfeasible in certain applications. Taking these factors into account, this paper focuses on index-free subsequence matching algorithms using DTW as the similarity measurement.

DTW is hard to compute because of the need of finding the optimal local alignment between time series. Therefore, DTW lower bounds [13, 15, 31] are widely utilized to relieve the heavy computational burden. The state-of-the-art algorithm in this field is UCR-Suite [23] and its variants [9, 27], which is well-known for its carefully designed cascading filtering strategy to discard candidates with the help of DTW lower bounds. However, the DTW lower bounds between the query series Q and a candidate (i.e., an m-length subsequence of the series S) require $O(m)$ time in most cases, where m is the length of the query Q. The overhead of verifying the DTW lower bounds for $O(n)$ candidates has become a critical bottleneck, especially for large m.

Proceedings of the VLDB Endowment, Vol. 18, No. 10 ISSN 2150-8097.
doi:10.14778/3748191.3748220

This paper addresses the above challenge by designing algorithms that cost only $O(\log m)$ time for lower bound of each candidate. Experiments show that the proposed lower bound can be several times faster than existing lower bounds even for short series. Considering m stands for the length of the query series, which can be up to tens of thousands, the proposed fast lower bound demonstrates considerable acceleration for the subsequence matching problem.

Based on the above improvements, this paper proposes an algorithm for subsequence matching named FSMDTW, which adopts the cascading filtering strategy and takes advantage of the proposed fast DTW lower bound. In addition, FSMDTW improves the effectiveness of LB_{Keogh} with little extra cost by leveraging the characteristics of the subsequence matching problem. Furthermore, FSMDTW promotes the efficiency of $LB_{Petitjean}$ by computing the projection vector on demand. Experiments on both real and artificial datasets demonstrate that the proposed FSMDTW algorithm is about 2.6 times faster than SOTA for short and medium length queries. For longer query sequences, FSMDTW outperforms SOTA by up to one order of magnitude.

The key contributions of this paper are summarized as follows.

1. This paper designs new efficient algorithms to compute DTW lower bounds for subsequence matching problem. To our knowledge, this is the first algorithm that computes DTW lower bounds with an average cost of $\Theta(\log m)$. The proposed algorithm is much faster than the existing lower bounds that cost $O(m)$ time, while it is still able to discard around 98% candidates. Considering m is the length of the query, this brings a significant advantage in performance of subsequence matching algorithms.

2. A refinement of LB_{Keogh} has been proposed, which is named as LB_{KE}. LB_{KE} takes advantage of a precomputed distance table based on the information of Q. LB_{KE} is tighter than LB_{Keogh}, but can be computed with little extra burden in the subsequence matching problem.

3. Based on the above advancements, this paper designs the FSMDTW algorithm, which addresses the subsequence matching problem by carefully leveraging the cascading filtering strategy and caching shared variables. In addition, FSMDTW also improves the efficiency of $LB_{Petitjean}$ by computing its projection vector and envelopes on demand.

4. The experiments carried out on both real and synthetic datasets show that the proposed FSMDTW algorithm significantly outperforms SOTA, especially for long queries. The proposed algorithm is about 2.6 times faster than the SOTA on short and medium-length queries and is up to a magnitude faster than SOTA on longer queries.

2 PROBLEM DEFINITION AND RELATED WORK

2.1 Definitions Used in This Paper

Definition 1. (Time Series) In this paper, a time series $S \in \mathbb{R}^n$ of length n is formalized as a series of n real numbers, i.e., $S = (s_1, s_2, s_3, ...s_n)$.

Definition 2. (Subsequences of Time series) A subsequence is defined as any continuous segment within the time series. In particular, an m-length subsequence starts from the ith position of series $S = (s_1, s_2, s_3, ...s_n)$ is denoted as $S_{(i)} = (s_i, s_{i+1}, s_{i+2}, ...s_{i+m-1})$, where $1 \le i \le n - m + 1$. In this paper, an m-length subsequence of S is also referred to as a *candidate*.

Definition 3. (Dynamic Time Warping) The DTW between two series $X = (x_1, x_2, ..., x_m) \in \mathbb{R}^m$ and $Q = (q_1, q_2, ..., q_m) \in \mathbb{R}^m$ is defined as follows.

$$DTW(X, Q) = \sqrt{(x_1 - q_1)^2 + MIN^2}, \tag{1}$$

where MIN is the minimum value among the three options, that are $DTW(X', Q')$, $DTW(X', Q)$, and $DTW(X, Q')$, $X' = (x_2, x_3..., x_m)$ and $Q' = (q_2, q_3..., q_m)$. In particular, the distance between an arbitrary series and an empty series is defined as infinity. The restriction of the warping path is necessary to prevent pathological alignments and to ensure that the DTW distance reflects meaningful similarities between two series [13, 24, 29, 32]. We employ the most commonly used Sakoe-Chiba band [26], which constrains the maximum distortion of an element on the time axis should not exceed $w \in \mathbb{N}^*$. [1]

Finally, some frequently used notations in the rest of the paper are listed in Table 1.

Table 1: Frequently Used Notations

Symbol	Description
S	The long series to be searched $S = (s_1, s_2, .., s_n)$.
Q	The query series $Q = (q_1, q_2, ..., q_m)$.
$S_{(i)}$	The m-length subsequence of S starts at the ith position, i.e., $S_{(i)} = (s_i, s_{i+1}, s_{i+2}, \ldots, s_{i+m-1})$.
$\mathbb{U}^Q$	The upper envelope of Q. $\mathbb{U}^Q = (\mathbb{U}_1^Q, \mathbb{U}_2^Q, ..., \mathbb{U}_m^Q)$, where $\mathbb{U}_i^Q = max\{q_{i-w}, q_{i-w+1}, ..., q_{i+w}\}$.
$\mathbb{L}^Q$	The lower envelope of Q. $\mathbb{L}^Q = (\mathbb{L}_1^Q, \mathbb{L}_2^Q, ..., \mathbb{L}_m^Q)$, where $\mathbb{L}_i^Q = min\{q_{i-w}, q_{i-w+1}, ..., q_{i+w}\}$.

2.2 Problem Definition

The z-normalized form of a subsequence $S_{(i)}$ is denoted as $\hat{S}_{(i)}$ and calculated as follows:

$$\hat{S}_{(i)} = (\frac{S_{(i)}[1] - \mu_i}{\sigma_i}, \frac{S_{(i)}[2] - \mu_i}{\sigma_i}, ..., \frac{S_{(i)}[m] - \mu_i}{\sigma_i}), \tag{2}$$

where μ_i and σ_i represent the mean and standard deviation of $S_{(i)}$, respectively. The importance of z-normalization for real-valued subsequences has gained considerable acknowledgment in recent years [2, 3, 18, 21, 24, 32–34, 37]. Following the perspectives of previous studies, this paper uses $DTW(\hat{S}_{(i)}, Q)$ to measure the dissimilarity of candidate subsequence $S_{(i)}$ and query Q. [2]

[1] $\mathbb{N}^*$ represents the set of positive integers.

[2] Because Q can be easily z-normalized before query possessing, it is always safe to assume $Q = \hat{Q}$. This paper does not explicitly normalize the query sequence Q for the sake of brevity.

Definition 4. (Subsequence Matching Problem) Given an input series $S \in \mathbb{R}^n$, a query $Q \in \mathbb{R}^m$ and a similarity threshold $\epsilon > 0$, the subsequence matching problem is to find the subsequences of S that are similar to query Q, i.e., $\{S_{(i)} | DTW(\hat{S}_{(i)}, Q) \leq \epsilon\}$.

2.3 Related Works

2.3.1 The lower bounds for DTW. Dynamic Time Warping (DTW) has served as one of the most important similarity measurements in time series data in the past 50 years [26], including online boot detection [8], sensing time series classification [20], forecasting the spread of COVID-19 [28], etc. Unfortunately, theoretical result indicates that the unbounded DTW cannot be exactly computed in $O(m^{2-\delta})$ time for any constant $\delta > 0$ if the *Strong Exponential Time Hypothesis* is true [1]. This implies that we are unable to find any algorithm for bounded DTW with parameter w that provides a significantly better time complexity than $O(wm^{1-\delta})$ because the bounded DTW is a special case of DTW with Sakoe-Chiba band (where $w = m$). Therefore, achieving an algorithm with a time complexity significantly better than the existing $O(wm)$ time dynamic programming algorithm is nearly impossible.

Most algorithms do not compute DTW directly. Instead, they avoid computing DTW by utilizing the *lower bounds of DTW*. Specifically, these lower bounds are less computationally expensive than DTW and provide approximations that are guaranteed not to exceed the true DTW score. Consequently, it is no longer necessary to compute the exact DTW if the lower bounds exceed ϵ. In fact, the DTW lower bound can filter out most of the candidates, so the computation overhead of the DTW lower bound is important for the efficiency of subsequence matching algorithms.

DTW lower bounds of $O(1)$ time complexity. These lower bounds including LB_{Kim} [14] and LB_{KimFL}[3] [23] utilize only constant elements from a series of length m. For example, given series $X = (x_1, x_2, \ldots, x_m)$ and $Q = (q_1, q_2, \ldots, q_m)$, LB_{KimFL} is defined as :

$$LB_{KimFL}(X, Q) = \sqrt{(x_1 - q_1)^2 + (x_m - q_m)^2 + r_1^2 + r_2^2 + r_3^2 + r_4^2}, \tag{3}$$

where $r_1 = \min\{(x_2 - q_1)^2, (x_2 - q_2)^2, (x_1 - q_2)^2\}$, $r_2 = \min\{(x_{m-1} - q_m)^2, (x_{m-1} - q_{m-1})^2, (x_m - q_{m-1})^2\}$, $r_3 = \min\{(x_1 - q_3)^2, (x_2 - q_3)^2, (x_3 - q_3)^2, (x_3 - q_2)^2, (x_3 - q_1)^2\}$, $r_4 = \min\{(x_m - q_{m-2})^2, (x_{m-1} - q_{m-2})^2, (x_{m-2} - q_{m-2})^2, (x_{m-2} - q_{m-1})^2, (x_{m-2} - q_m)^2\}$. That is, only the first and the last three elements are utilized from a series of length m.

However, the effectiveness of these lower bounds is often poor because they consider only constant number of the elements in the series. In general, these lower bounds are able to benefit the subsequence matching algorithms in some cases, but they do not work well on long time series.

Recently, an algorithm [7] has been proposed to compute DTW lower bounds in $O(n^2)$ time for the motif discovery problem, where $O(n^2)$ DTW scores between $O(n)$ subsequences need to be compared, achieving an average time cost of $O(1)$. However, this algorithm is specifically tailored for the DTW-based motif discovery and cannot be transferred to the subsequence matching problem.

DTW lower bounds of $O(m)$ time complexity. The vast majority of DTW lower bounds belongs to this category [13, 16, 29, 31] because accessing all the elements of an m-length series requires $O(m)$ time. The most successful lower bound of $O(m)$ time complexity is LB_{Keogh} [13], which is applied in most works that involve DTW. Given $X = (x_1, x_2, ..., x_m)$ and $Q = (q_1, q_2, ..., q_m)$,

$$LB_{Keogh}(X, Q) = \sqrt{\sum_{i=1}^{m} \begin{cases} (\mathbb{U}_i^Q - x_i)^2, & x_i > \mathbb{U}_i^Q \\ (\mathbb{L}_i^Q - x_i)^2, & x_i < \mathbb{L}_i^Q \\ 0, & \mathbb{L}_i^Q \leq x_i \leq \mathbb{U}_i^Q \end{cases}}, \tag{4}$$

where $\mathbb{U}_i^Q = max\{q_{i-w}, q_{i-w+1}, ..., q_{i+w}\}$, $\mathbb{L}_i^Q = min\{q_{i-w}, q_{i-w+1}, ..., q_{i+w}\}$, and w is the parameter of Sakoe-Chiba band representing the maximum distance of alignment. Additionally, the series $\mathbb{U}^Q = (\mathbb{U}_1^Q, \mathbb{U}_2^Q, ..., \mathbb{U}_m^Q)$ and $\mathbb{L}^Q = (\mathbb{L}_1^Q, \mathbb{L}_2^Q, ..., \mathbb{L}_m^Q)$ are referred as the upper envelope and the lower envelope of Q respectively. In summary, these lower bounds achieve good effectiveness, but they come with a higher time cost.

This paper introduces efficient algorithms that calculate DTW lower bounds with an average cost of $O(\log m)$. To the best of our knowledge, this is the first algorithm that computes DTW lower bound with such time complexity, achieving efficiency close to $O(1)$ time lower bounds while still maintaining satisfactory effectiveness with respect to $O(m)$ time lower bounds.

2.3.2 Subsequence Matching Algorithms Based on Dynamic Time Warping. Subsequence matching using DTW is a significant problem. However, the requirement for z-normalization and the uncertainty of the query length m present challenges for a perfect index-based algorithm [23]. ULISSE [17] only works when m falls within a predefined range determined by the index. KV-match [32] and the algorithm proposed in [6] limit the scale of z-normalization, which must be meticulously determined, as an excessively narrow range can result in missed query results, while a broad normalization range can significantly undermine efficiency.

Moreover, these index-based approaches require relatively expensive prepossessing steps, which involve additional time and space overhead before queries can be executed. The time to construct these indexes may take several hours, depending on the configuration. Furthermore, establishing such an index is not practical in streaming data processing [11, 22], therefore, it is necessary to investigate index-free subsequence matching algorithms.

This paper focuses on designing an efficient index-free algorithm. The index-free SOTA solution of the subsequence matching problem is UCR-Suite [23] and its variant UCR-MON Suite [9], which utilize LB_{KimFL} and LB_{Keogh} to discard the candidates.

3 COMPUTING DTW LOWER BOUNDS IN AVERAGE $O(\log m)$ TIME

This section introduces a lower bound for DTW named LB_{M+}, along with efficient algorithms to compute LB_{M+} in subsequence

[3] $LB_{KimFL}(X, Q)$ is originally defined as $\sqrt{(x_1 - q_1)^2 + (x_m - q_m)^2}$. Our definition is consistent with the variant implemented in the UCR-Suite. (https://www.cs.ucr.edu/~eamonn/UCRsuite.html)

matching problem. In this section, we temporally concentrate on a ℓ-length segment of S, denoted as T, and compute the lower bounds between Q every subsequence in $\{T_{(i)}\}_{i=1}^{\ell-m+1}$. Section 3.1 formalizes the definition of LB_{M+} and proves its correctness. Section 3.2 presents the basic idea for computing LB_{M+} in average $O(\log m)$ time. After that, Section 3.3 and 3.4 formally present the efficient algorithms to compute LB_{M+}. Finally, Section 3.5 discusses refining the effectiveness of LB_{M+} by selecting the mask vector.

3.1 Definition and Proof of Correctness for the Proposed DTW Lower bound

3.1.1 Overview of LB_{M+}. As shown in Figure 2, given any $X = (x_1, x_2, \ldots, x_m)$ and $Q = (q_1, q_2, \ldots, q_m)$, a matrix of size $m \times m$ can be constructed by taking the elements of X and Q as horizontal and vertical coordinates, respectively. A warping path is defined as the path in the matrix that starts from (x_1, q_1) (bottom left corner) and ends at (x_m, q_m) (top right corner) without crossing gray cells. According to Definition 3, computing $DTW(X, Q)$ equals finding a warping path such that the total costs of the cells that it passed through is minimized, where the cost of passing through cell (x_i, q_j) is defined as $(x_i - q_j)^2$.

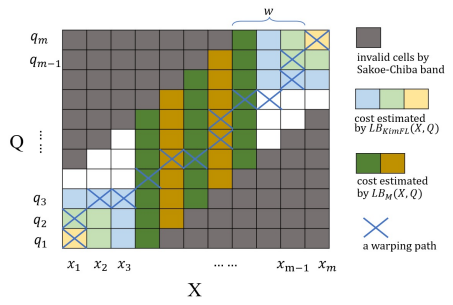


Figure 2: Illustration of a warping path and LB_{M+}.

The non-gray areas are cut into several bands which are distinguished by different colors in Figure 2. Any warping path must cross every of these bands, paying at least the lowest cost for cells within that band. Obviously, the smaller the size of a band, the tighter the lower bound will be. LB_{M+} basically uses the LB_{KimFL} lower bound on the first and last three data elements and adopts LB_M [7] as the lower bound for the middle part. Therefore, LB_{KimFL} provides a tighter lower bound at the beginning and end of a sequence compared to the general distance lower bound (i.e., vertical bands), making LB_{M+} practically tighter than LB_M.

3.1.2 Definition of LB_{M+}. Given query $Q = (q_1, q_2, \ldots, q_m) \in \mathbb{R}^m$, an m-length subsequence $X = (x_1, x_2, \ldots, x_m) \in \mathbb{R}^m$, and any vector $M \in \{0, 1\}^m$, LB_{M+} is defined as Equation (5):

$$LB_{M+}(X, Q) = \sqrt{\frac{[\max\{\sqrt{2p_2 - p_1} - \sqrt{p_1}, 0\}]^2}{4} + LB_{KimFL}(X, Q)^2}, \tag{5}$$

where

$$p_1 = \sum_{\{i|4 \le i \le m-3, M[i]=1\}} (\mathbb{U}_i^Q - \mathbb{L}_i^Q)^2, \tag{6}$$

$$p_2 = \sum_{\{i|4 \le i \le m-3, M[i]=1\}} [(x_i - \mathbb{L}_i^Q)^2 + (x_i - \mathbb{U}_i^Q)^2] \tag{7}$$

and $\mathbb{U}_i^Q$ and $\mathbb{L}_i^Q$ are the elements in envelopes of Q defined in Table 1.

3.1.3 Proof on Correctness of LB_{M+}.

Theorem 5. *Given any $Q \in \mathbb{R}^m$, $X \in \mathbb{R}^m$, and $M \in \{0, 1\}^m$ that satisfies $p_2 \ge p_1$,*

$$LB_{M+}(X, Q) \le DTW(X, Q). \tag{8}$$

Proof. Given any $X \in \mathbb{R}^m$ and $Q \in \mathbb{R}^m$, summing the minium costs of each band illustrated in Figure 2 implies,

$$DTW(X, Q)^2 \ge LB_{KimFL}(X, Q)^2 + \sum_{i=4}^{m-3} d_i^2, \tag{9}$$

where,

$$d_i = \min_{i-w \le j \le i+w} \left\{ (x_i - q_j)^2 \right\}. \tag{10}$$

According to [7],

$$\sqrt{\sum_{i=4}^{m-3} d_i^2} \ge \frac{\sqrt{2p_2 - p_1} - \sqrt{p_1}}{2} \tag{11}$$

If $p_2 \ge p_1$, then $\sqrt{2p_2 - p_1} - \sqrt{p_1} \ge 0$, therefore,

$$\sum_{i=4}^{m-3} d_i^2 \ge \frac{[\sqrt{2p_2 - p_1} - \sqrt{p_1}]^2}{4} \tag{12}$$

□

It should be noted that the cost of directly computing LB_{M+} defined as Equation (5) is still $O(m)$ time for each candidate. To achieve an average cost of $O(\log m)$, it is necessary to carefully design an algorithm that makes use of the characteristics of the subsequence matching problem.

3.2 Basic Idea for Computation LB_{M+} Efficiently in Subsequence Matching Problem

3.2.1 The core idea. The idea for computing LB_{M+} efficiently is as follows. Firstly, LB_{M+} is transformed into a series of inner products with mathematical derivation. Then, the inner products are computed using the Fast Fourier Transform (FFT). Leveraging the fact that candidate subsequences overlap with each other in the subsequence matching problem, the second step incurs only $O(logm)$ per inner product, and the first step only takes a constant time. To express sequence operations in a more concise way, we introduce the notations shown in Table 2.

3.2.2 Reducing LB_{M+} into Inner Products. Given the candidate subsequence $T_{(i)}$ and query Q, we convert $LB_{M+}(\hat{T}_{(i)}, Q)$ into several inner products. According to Equitation (5) and (6), to calculate $LB_{M+}(\hat{T}_{(i)}, Q)$, we only need the value of p_1, p_2 and $LB_{KimFL}(\hat{T}_{(i)}, Q)$. p_1 depends only on query Q, thus, its value can be shared by any $i \in \{1, 2, \ldots, \ell - m + 1\}$. Therefore, the cost of obtaining p_1 can be seen as a constant. In addition, LB_{KimFL} costs only $O(1)$ time. Therefore, the key to compute $LB_{M+}(\hat{T}_{(i)}, Q)$ depends on p_2. For the sake of clarity, we define $M' \in \{0, 1\}^m$ as

Table 2: The Notations Used in Section 3

Symbol	Description
$\oplus$	The element-wise addition between series, i.e., $X \oplus Q = (x_1 + q_1, x_2 + q_2, \ldots, x_m + q_m)$ for any $X = (x_1, x_2, \ldots, x_m) \in \mathbb{R}^m, Q = (q_1, q_2, \ldots, q_m) \in \mathbb{R}^m$.
$\ominus$	The element-wise minus between series, i.e., $X \ominus Q = (x_1 - q_1, x_2 - q_2, \ldots, x_m - q_m), X, Q \in \mathbb{R}^m$.
$\odot$	The element-wise product between series, i.e., $X \odot Q = (x_1 q_1, x_2 q_2, \ldots, x_m q_m)$.
$\langle X, Q \rangle$	The inner product between series X and Q.

$$M'[i] = \begin{cases} M[i], & 4 \le i \le m-3 \\ 0, & otherwise. \end{cases} \tag{13}$$

By Equation(7), p_2 can be decomposed as

$$\begin{aligned} p_2 &= \sum_{\{j|M'[j]=1\}} \left[\left(\frac{t_{i+j-1} - \mu_i}{\sigma_i} - \mathbb{L}_j^Q \right)^2 + \left(\frac{t_{i+j-1} - \mu_i}{\sigma_i} - \mathbb{U}_j^Q \right)^2 \right] \\ &= \sum_{\{j|M'[j]=1\}} [\frac{2}{\sigma_i^2} t_{i+j-1}^2 - \frac{4\mu_i}{\sigma_i^2} t_{i+j-1} - \frac{2}{\sigma_i} (t_{i+j-1} \mathbb{U}_j^Q) \\ &\quad - \frac{2}{\sigma_i} (t_{i+j-1} \mathbb{L}_j^Q) + \frac{2\mu_i^2}{\sigma_i^2} + \frac{2\mu_i}{\sigma_i} (\mathbb{L}_j^Q + \mathbb{U}_j^Q) + \left(\mathbb{U}_j^Q \right)^2 + \left(\mathbb{L}_j^Q \right)^2], \end{aligned} \tag{14}$$

where $\mathbb{U}^Q = (\mathbb{U}_1^Q, \mathbb{U}_2^Q, ..., \mathbb{U}_m^Q)$, $\mathbb{L}^Q = (\mathbb{L}_1^Q, \mathbb{L}_2^Q, ..., \mathbb{L}_m^Q)$ are the envelops of Q defined in Table 1, and μ_i, σ_i are the mean and standard deviation of candidate subsequence $T_{(i)}$.

Fortunately, all terms in Equation (18) can be efficiently computed in the subsequence matching problem for $i \in \{1, 2, \ldots, \ell - m + 1\}$. Because $M' \in \{0, 1\}^m$, summing the terms according to M', is equivalent to computing the inner product. For example,

$$\sum_{\{j|M'[j]=1\}} t_{i+j-1} = \sum_{\{1 \le j \le m\}} t_{i+j-1} * M'[j] \quad = \langle T_{(i)}, M' \rangle, \tag{15}$$

where $\langle T_{(i)}, M' \rangle$ is the inner product of $T_{(i)} \in \mathbb{R}^m$ and M' is defined in Table 2. Let "$\odot$" be the element-wise product defined in Table 2, we have

$$\sum_{\{j|M'[j]=1\}} t_{i+j-1}^2 = \langle (T_{(i)} \odot T_{(i)}), M' \rangle. \tag{16}$$

Besides,

$$\sum_{\{j|M'[j]=1\}} t_{i+j-1} \mathbb{U}_j^Q = \langle T_{(i)}, (M' \odot \mathbb{U}^Q) \rangle. \tag{17}$$

Following the same methodology, $\sum_{\{j|M'[j]=1\}} t_{i+j-1} \mathbb{L}_j^Q = \langle T_{(i)}, (M' \odot \mathbb{L}^Q) \rangle$, $\sum_{\{j|M'[j]=1\}} \left(\mathbb{L}_j^Q \right)^2 = \langle (\mathbb{L}^Q \odot \mathbb{L}^Q), M' \rangle$ and $\sum_{\{j|M'[j]=1\}} \left(\mathbb{U}_j^Q \right)^2 = \langle (\mathbb{U}^Q \odot \mathbb{U}^Q), M' \rangle$ can also be transformed into inner products between $M', \mathbb{U}^Q, \mathbb{L}^Q$ and $T_{(i)}$.

Consequently, we have

$$\begin{aligned} p_2 &= \frac{2}{\sigma_i^2} \langle (T_{(i)} \odot T_{(i)}), M' \rangle - \frac{4\mu_i}{\sigma_i^2} \langle T_{(i)}, M' \rangle - \frac{2}{\sigma_i} \langle T_{(i)}, (M' \odot \mathbb{U}^Q) \rangle \\ &\quad - \frac{2}{\sigma_i} \langle T_{(i)}, (M' \odot \mathbb{L}^Q) \rangle + \frac{2\mu_i^2}{\sigma_i^2} \langle M', M' \rangle + \frac{2\mu_i}{\sigma_i} \langle (\mathbb{L}^Q \oplus \mathbb{U}^Q), M' \rangle \\ &\quad + \langle (\mathbb{L}^Q \odot \mathbb{L}^Q), M' \rangle + \langle (\mathbb{U}^Q \odot \mathbb{U}^Q), M' \rangle. \end{aligned} \tag{18}$$

So far, $LB_{M+}(\hat{T}_{(i)}, Q)$ has been decomposed into calculations of inner products, as well as other operations with constant time complexity. These inner products can be computed in batches using the Fourier transform, with the average cost of each inner product being only logarithmic. Therefore, the average cost of $LB_{M+}(\hat{T}_{(i)}, Q)$ can be reduced to $O(\log m)$ when $\ell = O(m)$.

3.2.3 The Sliding Inner Product Algorithm (SIP). Given any $T = (t_1, t_2, t_3, \ldots t_\ell) \in \mathbb{R}^\ell$ and query $Q \in \mathbb{R}^m$. Let $T_{(i)} = (t_i, t_{i+1}, \ldots, t_{i+m-1})$ and $< T_{(i)}, Q >$ be the inner product of $T_{(i)}$ and Q. Algorithm 1 describes the subroutine to compute $\{< T_{(i)}, Q >\}_{i=1}^{\ell-m+1}$, the inner products between Q and every m-length subsequence of T, in $O(\ell \log \ell)$ time. The inner products between Q and $T_{(1)}, \ldots, T_{(\ell-m+1)}$ can be regarded as the result of a convolution between Q and T. Therefore, they can be computed with Discrete Fourier Transform (DFT) and Inverse Discrete Fourier Transform (IDFT) in $O(\ell \log \ell)$ time [30, 35].

Algorithm 1: *SIP*(T, X)

Input: $T \in \mathbb{R}^\ell$, $X \in \mathbb{R}^m$, $(\ell > m)$.
Output: **The inner product between** X **and** $T_{(i)}$ **for** $i \in \{1, 2, \ldots, \ell - m + 1\}$.

1 $X_{pad} \leftarrow$ pad X with $\ell - m + 1$ zeros;
2 $X_{pad} \leftarrow$ inverse X_{pad};
3 $T_{DFT} \leftarrow DFT(T)$; `// Fast Fourier Transform`
4 $X_{DFT} \leftarrow DFT(X_{pad})$;
5 $COV \leftarrow IDFT(T_{DFT} \odot X_{DFT})$;
6 **return** the real part of $COV[m-1 : \ell]$.

3.3 Computing $LB_{M+}(\hat{T}_{(i)}, Q)$ in Average $O(\log m)$ Time

Now, we formally present the algorithm for calculating LB_{M+} based on the ideas discussed in Section 3.2.

3.3.1 Algorithm Overview. The pseudocode for computing $LB_{M+}(\hat{T}_{(i)}, Q)$ is described as Algorithm 2, where " $\ominus$", "$\odot$", and "$\oplus$" are the element-wise operations defined in Table 2. Firstly, Lines (1-5) initialize the mask vector M. It should be noted that Algorithm 2 is correct for any mask vector $M \in \{0, 1\}^\ell$, that is, $lbQ_i \le DTW(\hat{T}_{(i)}, Q)$. However, M affects the proximity of lbQ_i to $DTW(\hat{T}_{(i)}, Q)$. The selection of M will be further discussed in Section 3.5.

Then, Lines (6-7) compute the mean value, $\{\mu_i\}_{i=1}^{\ell-m+1}$, and the standard deviation, $\{\sigma_i\}_{i=1}^{\ell-m+1}$, within every m-length sliding window of T. The corresponding algorithms to compute the moving

average and the moving standard deviation in $O(\ell)$ time are denoted as $MVStd(T, m)$ and $MVMean(T, m)$ respectively[4]. According to algebraic knowledge, given any series $T = \{t_1, t_2, \ldots, t_\ell\} \in \mathbb{R}^\ell$, the mean, sum, or standard deviation of the m elements in the sliding window can be obtained from the first-order central moment $\sum_{i=k}^{k+m-1} t_i$ and the second-order central moment $\sum_{i=k}^{k+m-1} (t_i)^2$ of the elements, where k is the starting position of the sliding window. For space limitations, this paper does not list the corresponding pseudocode, we recommend the readers refer to [24] for details.

Algorithm 2: $LB\_Q(Q, T)$

Input: Query $Q \in \mathbb{R}^m$ and ℓ-length series $T \in \mathbb{R}^\ell$.
Output: The Lower Bound of $D(Q, T_{(i)})$, $\{lbQ_i\}_{i=1}^{\ell-m+1}$.

1 $\mathbb{U}^Q, \mathbb{L}^Q \leftarrow$ the lower and upper envelopes of Q;
2 $M \leftarrow \{0\}^m$; `// M is initialized with m zeros`
3 **foreach** $4 \le i \le m-3$ **do**
4 **if** $\Phi(\mathbb{U}_i^Q) - \Phi(\mathbb{L}_i^Q) \le \frac{1}{2}$ **then**
5 $M[i] \leftarrow 1$;
6 $\{\mu_i\}_{i=1}^{\ell-m+1} \leftarrow$ ***MVMean(T,m)*** ; `// O(ℓ) time`
7 $\{\sigma_i\}_{i=1}^{\ell-m+1} \leftarrow$ ***MVStd(T,m)*** ; `// O(ℓ) time`
8 $UM \leftarrow M \odot \mathbb{U}^Q$; $LM \leftarrow M \odot \mathbb{L}^Q$;
9 $UU \leftarrow \mathbb{U}^Q \odot \mathbb{U}^Q$; $LL \leftarrow \mathbb{L}^Q \odot \mathbb{L}^Q$; $TT \leftarrow T \odot T$;
10 $\{TM_i\}_{i=1}^{\ell-m+1} \leftarrow$ ***SIP***(T, M) ; `// Algorithm 1`
11 $\{TTM_i\}_{i=1}^{\ell-m+1} \leftarrow$ ***SIP***(TT, M) ;
12 $\{TUM_i\}_{i=1}^{\ell-m+1} \leftarrow$ ***SIP***(T, UM) ;
13 $\{TLM_i\}_{i=1}^{\ell-m+1} \leftarrow$ ***SIP***(T, LM) ;
14 $D_{UL} \leftarrow ED(\mathbb{U}^Q \odot M, \mathbb{L}^Q \odot M)$;
15 $c_1 \leftarrow \langle UU, M \rangle$; $c_2 \leftarrow \langle LL, M \rangle$;
16 $c_3 \leftarrow \langle \mathbb{U}^Q, M \rangle$; $c_4 \leftarrow \langle \mathbb{L}^Q, M \rangle$; $c_5 \leftarrow \langle M, M \rangle$;
17 **foreach** $1 \le i \le \ell - m + 1$ **do**
18 $D_{UX} \leftarrow$ $c_1 - \frac{2}{\sigma_i}[TUM_i - \mu_i c_3] + \frac{1}{\sigma_i^2}\left[TTM_i - 2\mu_i TM_i + \mu_i^2 c_5\right]$;
19 $D_{LX} \leftarrow$ $c_2 - \frac{2}{\sigma_i}[TLM_i - \mu_i c_4] + \frac{1}{\sigma_i^2}\left[TTM_i - 2\mu_i TM_i + \mu_i^2 c_5\right]$;
20 $lbQ_i \leftarrow \max\{\frac{1}{2}\left[\sqrt{2D_{UX} + 2D_{LX} - D_{UL}^2} - D_{UL}\right], 0\}$;
21 **if** $lbQ_i \le \epsilon$ **then**
22 $lbQ_i \leftarrow \sqrt{(lbQ_i)^2 + LB_{KimFL}(Q, \hat{T}_{(i)})^2}$;
23 Set $M \leftarrow \{1\}^\ell$ and repeat procedure in Lines (8-22) to compute $\{lbQ'_i\}_{i=1}^{\ell-m+1}$;
24 **foreach** $1 \le i \le \ell - m + 1$ **do**
25 $lbQ_i \leftarrow max\{lbQ_i, lbQ'_i\}$;
26 **return** lbQ_i for $i \in \{1, 2, 3, ..., \ell - m + 1\}$;

Lines (10-13) invoke Algorithm 1 to calculate the inner products. Finally, Lines (17-22) compute $LB_{M+}(\hat{T}_{(i)}, Q)$ according to Equation (5). There is no need to spend extra time on $LB_{KimFL}(\hat{T}_{(i)}, Q)$ if lbQ_i exceeds the similarity threshold ϵ (Lines 24-25). Additionally, we repeat Lines (8-22) using $\{1\}^\ell$ as the mask vector to compute lbQ'_i and output $max\{lbQ_i, lbQ'_i\}$.

[4]Another variant of $MVMean(T, m)$ is $MVSum(T, m)$, which computes the moving sum of the sliding window.

3.3.2 Algorithm Analysis. We now show that when the length of T, denoted as ℓ, satisfies $\ell \ge 2m$, the average cost of Algorithm 2 to compute LB_{M+} for each candidate is only $O(\log m)$.

Theorem 6. *The time complexity of Algorithm 2 is $O(\ell \log \ell)$.*

Proof. Line (1) computes the envelopes of Q with monotonic queue and costs $O(m)$. Lines (2-5) of algorithm 2 compute the mask vector $M \in \mathbb{R}^m$ by checking the cumulative probability distribution of the standard normal distributions through a lookup table and cost $O(m)$ time. Then, Lines (6-7) compute the mean and standard deviation for each candidate and cost $O(\ell)$ time. Lines (8-16) invoke Algorithm 1 to compute a series of inner products, using $O(\ell \log \ell)$ time. Finally, Lines (17-22) iterate at most $\ell - m + 1$ times, with a constant time cost for each iteration. Therefore, the time complexity of this part is still $O(\ell)$. Finally, Lines (23-25) change the mask vector and repeat the above subroutine. Therefore, the time complexity of the algorithm remains unchanged under the big O notation. In summary, the time complexity is $O(\ell \log \ell)$. □

Note that T is a segment of the time series $S \in \mathbb{R}^n$, and its length ℓ can be dynamically determined according to m. We set $\ell = \lceil Km \rceil$, where $K > 2$ is a constant. Algorithm 2 computes DTW lower bounds for $\ell - m + 1$ candidates within $O(\ell \log \ell)$ time. That is, the average time to compute each lower bound is $O(\frac{\ell \log \ell}{\ell - m + 1}) = O(\frac{\lceil Km \rceil \log(\lceil Km \rceil)}{\lceil (K-1)m \rceil + 1}) = O(\log m)$.

3.4 Computing $LB_{M+}(Q, \hat{T}_{(i)})$ in Average $O(\log m)$ Time

LB_{M+} is asymmetric with respect to the input, i.e., $LB_{M+}(\hat{T}_{(i)}, Q) \ne LB_{M+}(Q, \hat{T}_{(i)})$, as illustrated in Figure 3. To ensure that as many potential candidates as possible are filtered out, we need to compute $LB_{M+}(Q, \hat{T}_{(i)})$.

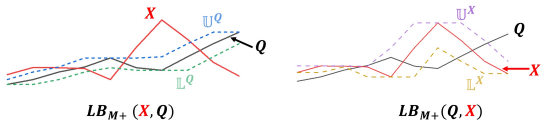


Figure 3: Asymmetry of the DTW lower bounds. Given $X, Q \in \mathbb{R}^m$, $LB_{M+}(X, Q)$ computes DTW lower bound according to $X, \mathbb{U}^Q$ and $\mathbb{L}^Q$ (left). $LB_{M+}(Q, X)$ computes DTW lower bound according to $Q, \mathbb{U}^X$ and $\mathbb{L}^X$ (right). Therefore, $LB_{M+}(X, Q) \ne LB_{M+}(Q, X)$ in the vast majority of cases.

Therefore, we still need to compute $LB_{M+}(Q, \hat{T}_{(i)})$ for $i \in \{1, 2, \ldots, \ell - m + 1\}$. However, directly reusing Algorithm 2 simply by exchanging T and Q is not feasible for the following reasons: 1) $T \in \mathbb{R}^\ell$ and $Q \in \mathbb{R}^m$ are of different lengths. 2) The candidate subsequences $\{T_{(i)}\}_{i=1}^{i=\ell-m+1}$ need to be normalized whereas Q does not.

The detailed procedure to compute $LB_{M+}(Q, \hat{T}_{(i)})$ is described as Algorithm 3. The detailed derivation and proof of correctness are omitted due to space limitation. Similarly, $LB_{M+}(Q, \hat{T}_{(i)})$ is transformed into several inner products. These inner products are calculated using algorithm *SIP*. Therefore, the time complexity of

this algorithm remains $O(\ell \log \ell)$, and the overhead to compute the DTW lower bound for each candidate is still $O(\log m)$ by setting the value of $\ell \geq 2m$.

An important difference between Algorithm 2 and Algorithm 3 is their mask vectors. In Algorithm 2, the mask vector is computed based on the envelope of Q, while in Algorithm 3 it is calculated based on the upper and lower envelopes of T. Therefore, there are significant differences in the length and calculation of the mask vector between the two algorithms, which will be discussed in Section 3.5.

Algorithm 3: $LB\_T(Q, T)$

Input: **Query** $Q \in \mathbb{R}^m$, **series** $T \in \mathbb{R}^\ell$.
Output: **The lower bound of** $D(Q, T_{(i)})$, $\{lbT_i\}_{i=1}^{\ell-m+1}$.

$\mathbb{U}^T, \mathbb{L}^T \leftarrow$ the lower and upper envelopes of T;
$M \leftarrow$ ***MASK_T(Q,T)*** ; // Algorithm 4
$\{\mu_i\}_{i=1}^{\ell-m+1} \leftarrow$***MVMean(T,ℓ)*** ; $\{\sigma_i\}_{i=1}^{\ell-m+1} \leftarrow$***MVStd(T,ℓ)*** ;
$UM \leftarrow M \odot \mathbb{U}^T$; $LM \leftarrow M \odot \mathbb{L}^T$; $QQ \leftarrow Q \odot Q$;
$\{QM_i\}_{i=1}^{\ell-m+1} \leftarrow$ ***SIP***(M, Q) ; // Algorithm 1
$\{QQM_i\}_{i=1}^{\ell-m+1} \leftarrow$ ***SIP***(M, QQ) ;
$\{ULQM_i\}_{i=1}^{\ell-m+1} \leftarrow$ ***SIP***$(UM \oplus LM, Q)$;
$\{a_i\}_{i=1}^{\ell-m+1} \leftarrow$ ***MVSum***$((UM \ominus LM) \odot (UM \ominus LM), m)$;
$\{b_i\}_{i=1}^{\ell-m+1} \leftarrow$ ***MVSum***$((UM \odot UM) \oplus (LM \odot LM), m)$;
$\{c_i\}_{i=1}^{\ell-m+1} \leftarrow$ ***MVSum***$(UM \oplus LM, m)$;
$\{d_i\}_{i=1}^{\ell-m+1} \leftarrow$ ***MVSum***$(M \odot M, m)$;
foreach $1 \leq i \leq \ell - m + 1$ **do**
  $D_{UL} \leftarrow \sqrt{a_i}/\sigma_i$;
  $D_{ULX} \leftarrow \frac{1}{\sigma_i^2}(b_i - 2\mu_i c_i + 2\mu_i^2 d_i) - \frac{2}{\sigma_i}(ULQM_i - 2\mu_i QM_i) + 2QQM_i$;
  $lbT_i \leftarrow \max\left\{\frac{1}{2}\left[\sqrt{2D_{ULX} - D_{UL}^2} - D_{UL}\right], 0\right\}$;
  if $lbT_i \leq \epsilon$ **then**
    $lbT_i \leftarrow \sqrt{(lbT_i)^2 + LB_{KimFL}(\hat{T}_{(i)}, Q)^2}$;
return lbT_i for $i \in \{1, 2, 3, ..., \ell - m + 1\}$;

3.5 Selecting Mask Vectors

To improve the efficiency of subsequence matching problems, it is desirable that the value of LB_{M+} is as large as possible. The mask vector M does not affect the time complexity or the correctness of Algorithm 2 and 3. However, M influences the value of LB_{M+}. Now it is time to explore how to choose M according to the data distribution to ensure the effectiveness of LB_{M+}.

3.5.1 The Impact of Masked Vector on the Effectiveness of LB_{M+}. First, consider a simple case where there is only a candidate X (without considering the impact of sequence normalization) and a query sequence Q. Intuitively reviewing Equation (5) ~ (7), $LB_{M+}(X, Q)$ selectively computes the lower bound of DTW only with a subset of the elements in X and Q. The actual role of the mask vector M is to mark the index of this subset. That is, $LB_{M+}(X, Q)$ uses the ith elements of X,Q, $\mathbb{L}^Q$ and $\mathbb{U}^Q$ (i.e. x_i, q_i, $\mathbb{L}_i^Q$ and $\mathbb{U}_i^Q$) if and only if $M[i] = 1$ or $i \in \{1, 2, 3, m-2, m-1, m\}$. Figure 4 shows the example where the mask vector M = 00001001111000000. This means that only the elements in the brown dashed rectangle of the figure are used to calculate the DTW lower bound (if we ignore the first three elements and the last three elements).

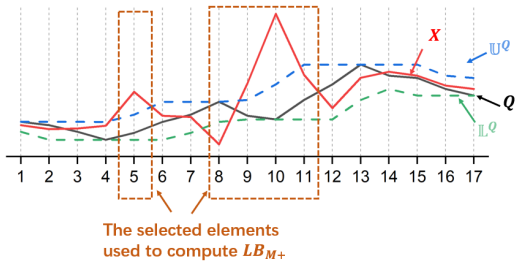


Figure 4: Illustration of $LB_{M+}(X, Q)$. The elements in the brown dashed boxes are selected to compute $LB_{M+}(X, Q)$.

A less intuitive fact is that adding more elements to this subset determined by M may undermine the effectiveness of LB_{M+}, degrading LB_{M+} to LB_{KimFL} in the worst case. Subsection 3.5.2 will quantitatively analyze when this occurs. An important observation is that the elements in series contribute variously to the lower bound $LB_{M+}(X, Q)$. For example, elements in the brown dashed boxes contribute the most significant difference between X and Q in Figure 4. Intuitively, when the selected subset of elements exhibits significant differences, LB_{M+} is more likely to yield a better (larger) DTW lower bound. Consequently, we need a mask vector to enhance the effectiveness of the DTW lower bound by selecting elements that are likely to benefit the DTW lower bounds.

3.5.2 Principles for Selecting a Good Mask Vector. Given a candidate subsequence $X \in \mathbb{R}^m$ and query $Q = (q_1, q_2, \ldots, q_m)$, $M[i]$ should be set to 0 if $x_i \in [\mathbb{L}_i^Q, \mathbb{U}_i^Q]$, where $\mathbb{L}_i^Q, \mathbb{U}_i^Q$ are the ith elements in the envelopes of Q, as defined in Table 1. The specific derivation process is shown in Theorem 7.

Theorem 7. *Given any mask vector $M \in \{0, 1\}^m$ where excites $1 \leq t \leq m$ satisfies that $M[t] = 1$ and $x_t \in [\mathbb{L}_t^Q, \mathbb{U}_t^Q]$, inequality (19) holds true for any X and Q :*

$$LB_{M+}(X, Q) \leq LB'_{M+}(X, Q), \tag{19}$$

where $LB'_{M+}(X, Q)$ is computed by using M' as the mask vector and

$$M'[i] = \begin{cases} 0, & i = t \\ M[i], & others. \end{cases} \tag{20}$$

Proof. Denote γ_1 as

$$\gamma_1 = \sum_{\{i|4 \leq i \leq m-3, M[i]=1\}} [(x_i - \mathbb{L}_i^Q)^2 + (x_i - \mathbb{U}_i^Q)^2 - (\mathbb{U}_i^Q - \mathbb{L}_i^Q)^2]. \tag{21}$$

If $\gamma_1 < 0$, then $LB_{M+}(X, Q) = LB_{KimFL}(X, Q) \leq LB'_{M+}(X, Q)$ and the proof is complete.

Otherwise, we have $\gamma_1 \geq 0$. Obviously, $\mathbb{L}_t^Q \leq x_t \leq \mathbb{U}_t^Q$ implies,

$$\sum_{\{i|4 \leq i \leq m-3, M'[i]=1\}} [(x_i - \mathbb{L}_i^Q)^2 + (x_i - \mathbb{U}_i^Q)^2 - (\mathbb{U}_i^Q - \mathbb{L}_i^Q)^2] \geq 0. \tag{22}$$

Denote $r_1 = \sqrt{LB_{M+}(X,Q)^2 - LB_{KimFL}(X,Q)^2}$ and $r_2 = \sqrt{LB'_{M+}(X,Q)^2 - LB_{KimFL}(X,Q)^2}$. Inequality (22) means $r_2 \geq 0$, and $\gamma_1 \geq 0$ means $r_1 \geq 0$. According to the definition of $LB(X,Q)$ and M',

$$\begin{aligned}
&\left[2r_1 + \sqrt{\sum_{\{i|4\leq i\leq m-3, M[i]=1\}} (\mathbb{U}_i^Q - \mathbb{L}_i^Q)^2}\right]^2 - \\
&\left[2r_2 + \sqrt{\sum_{\{i|4\leq i\leq m-3, M'[i]=1\}} (\mathbb{U}_i^Q - \mathbb{L}_i^Q)^2}\right]^2 \\
&= 2(x_t - \mathbb{L}_t^Q)^2 + 2(x_t - \mathbb{U}_t^Q)^2 - (\mathbb{U}_t^Q - \mathbb{L}_t^Q)^2 \\
&\leq (\mathbb{U}_t^Q - \mathbb{L}_t^Q)^2.
\end{aligned} \tag{23}$$

Algebra transformation of Equation (23) indicates,

$$(r_1 - r_2)\left[r_1 + r_2 + \sqrt{\sum_{\{i|4\leq i\leq m-3, M[i]=1\}} (\mathbb{U}_i^Q - \mathbb{L}_i^Q)^2}\right] \leq 0. \tag{24}$$

Considering $r_1 \geq 0$ and $r_2 \geq 0$, we have $r_2 \geq r_1 \geq 0$. Consequently, the proof is done by the fact that.

$$LB^*(X,Q)^2 - LB(X,Q)^2 = (r_2)^2 - (r_1)^2 \geq 0 \tag{25}$$

□

As a direct corollary of Theorem 7, in any case where $x_i \in [\mathbb{L}_i^Q, \mathbb{U}_i^Q]$, the value of $M[i]$ should always be 0. Otherwise, there will always be a better mask vector M' that deterministically produces a DTW lower bound that is equal to or better than the one obtained with the origin mask vector M.

3.5.3 Computing mask vector for Algorithm 2. The average time budget to compute LB_{M+} in this section is only $O(logm)$. The problem is that applying the principle described in Theorem 7 to calculate the mask vector for each candidate costs $O(m)$ time. Therefore, we can not individually compute the mask vector for each candidate. Consequently, heuristic mask vectors need to be formulated separately for Algorithm 2 and Algorithm 3, respectively.

Algorithm 2 computes $LB_{M+}(\hat{T}_{(i)}, Q)$ with $\{\hat{T}_{(i)}\}_{i=1}^{\ell-m+1}$, $\mathbb{U}^Q$, and $\mathbb{L}^Q$. Since it is not possible to compute a separate mask vector for each candidate, we only use a single mask vector $M \in \{0,1\}^m$ shared by the calculation of $LB_{M+}(\hat{T}_{(i)}, Q)$ for $i \in \{1,2,\ldots,\ell-m+1\}$. Consequently, the value of $M[j]$ should be good for the majority of candidates. That is, following the above principle, $M[j] = 1$ if and only if, for more than half of the candidates, the jth elements $\hat{T}_{(i)}[j]$ do not fall into the interval $[\mathbb{L}_j^Q, \mathbb{U}_j^Q]$ $(1 \leq j \leq n)$.

Because subsequences $\{\hat{T}_{(i)}\}_{i=1}^{\ell-m+1}$ have been z-normalized, i.e. the elements of $\{\hat{T}_{(i)}\}_{i=1}^{\ell-m+1}$ follow some distribution with a mean of 0 and a standard deviation of 1. Therefore, Algorithm 2 assumes $\{\hat{T}_{(i)}[j]\}_{i=1}^{\ell-m+1}$ subject to the standard normal distribution for any fixed j. Let $\Phi(X)$ denote the cumulative distribution function (CDF) of the standard normal distribution, then $\Phi(\mathbb{U}_j^Q) - \Phi(\mathbb{L}_j^Q)$ presents the possibility that the elements in $\{\hat{T}_{(i)}[j]\}_{i=1}^{\ell-m+1}$ fall into $[\mathbb{L}_j^Q, \mathbb{U}_j^Q]$. Therefore, $M[j]$ is set to 1 if this possibility is less than $\frac{1}{2}$ in Lines (3-5) of Algorithm 2.

3.5.4 Computing mask vector for Algorithm 3. Algorithm 3 computes $LB_{M+}(Q, \hat{T}_{(i)})$ with Q, $\mathbb{U}^{\hat{T}_{(i)}}$ and $\mathbb{L}^{\hat{T}_{(i)}}$, which are the upper and lower envelopes of the normalize subsequence $\hat{T}_{(i)}$. The Algorithm 3 requires a mask vector $M \in \{0,1\}^\ell$ and computes $LB_{M+}(Q, \hat{T}_{(i)})$ using $(M[i], M[i+1], \ldots, M[i+m-1]) \in \mathbb{R}^m$ for each $1 \leq i \leq \ell - m + 1$.

The choice of the mask vector still follows the principle mentioned above. In the ideal case we should check if $q_{j-i} \in [\mathbb{L}_{j-i}^{\hat{T}_{(i)}}, \mathbb{U}_{j-i}^{\hat{T}_{(i)}}]$ for each $j - m < i \leq j$. We set $M[j] = 1$ if and only if it is good for computation of LB_{M+} of most of the candidates. Unfortunately, $\mathbb{U}^{\hat{T}_{(i)}}$ and $\mathbb{L}^{\hat{T}_{(i)}}$ cannot be precisely computed because z-normalizing all the candidates $\{\hat{T}_{(i)}\}_{i=1}^{\ell-m+1}$ requires $O(m\ell)$ time, which is unaffordable for our $O(\ell \log \ell)$ time budget. Therefore, we estimate the envelope interval for the jth element, $[\mathbb{L}_{j-i}^{\hat{T}_{(i)}}, \mathbb{U}_{j-i}^{\hat{T}_{(i)}}]$, with $[\frac{\mathbb{L}_j^T - \mu_{j-\lfloor m/2 \rfloor}}{\sigma_{j-\lfloor m/2 \rfloor}}, \frac{\mathbb{U}_j^T - \mu_{j-\lfloor m/2 \rfloor}}{\sigma_{j-\lfloor m/2 \rfloor}}]$. Algorithm 4 describes the above heuristic, where an equal-width histogram to estimate the CDF of Q (Lines 3-6). $M[j]$ is set to 1 if and only if the probability of $\{q_i\}_{i=1}^m$ falls into $[\frac{\mathbb{L}_i^T - \mu_{i-\lfloor m/2 \rfloor}}{\sigma_{i-\lfloor m/2 \rfloor}}, \frac{\mathbb{U}_i^T - \mu_{i-\lfloor m/2 \rfloor}}{\sigma_{i-\lfloor m/2 \rfloor}}]$ is less than $\frac{1}{2}$.

Algorithm 4: *MASK_T*(Q,T)

Input: **Query** $Q = (q_1, q_2, \ldots, q_m) \in \mathbb{R}^m$ **and segment** $T = (t_1, t_2, \ldots, t_\ell) \in \mathbb{R}^\ell$.
Output: **The mask vector** $M \in \{0,1\}^\ell$.

1 $\mathbb{U}^T, \mathbb{L}^T \leftarrow$ the lower and upper envelopes of T ;
2 $Q_{max}, Q_{min} \leftarrow$ the maximum and minimum elements in Q ;
3 **foreach** $i \in \{1,2,\ldots,m\}$ **do**
4 $\quad Bin[h(q_i)] += 1$; // $h(v) = \lfloor \frac{(v - Q_{min})|Bin|}{Q_{max} - Q_{min}} \rfloor$
5 **foreach** $i \in \{2,3,\ldots,|Bin|\}$ **do**
6 $\quad Bin[i] \leftarrow Bin[i-1] + Bin[i]$;
7 $M \leftarrow \{0\}^\ell$; // M is initialized with ℓ zeros
8 **foreach** $4 \leq i \leq \ell - 3$ **do**
9 $\quad t \leftarrow i - \lfloor m/2 \rfloor$;
10 $\quad$ **if** $\lfloor m/2 \rfloor + 1 \leq i \leq \ell - m + 1 + \lfloor m/2 \rfloor$ **then**
11 $\quad\quad u \leftarrow \frac{\mathbb{U}_{(i)}^T - \mu_t}{\sigma_t}; l \leftarrow \frac{\mathbb{L}_{(i)}^T - \mu_t}{\sigma_t}$;
12 $\quad$ **else if** $i \leq \lfloor m/2 \rfloor$ **then**
13 $\quad\quad u \leftarrow \frac{\mathbb{U}_{(i)}^T - \mu_1}{\sigma_1}; l \leftarrow \frac{\mathbb{L}_{(i)}^T - \mu_1}{\sigma_1}$;
14 $\quad$ **else**
15 $\quad\quad u \leftarrow \frac{\mathbb{U}_{(i)}^T - \mu_{\ell-m+1}}{\sigma_{\ell-m+1}}; l \leftarrow \frac{\mathbb{L}_{(i)}^T - \mu_{\ell-m+1}}{\sigma_{\ell-m+1}}$;
16 $\quad$ **if** $\frac{Bin[h(u)] - Bin[h(l)]}{m} \leq \frac{1}{2}$ **then**
17 $\quad\quad M[i] \leftarrow 1$;
18 **return** M;

4 FSMDTW: A FAST SUBSEQUENCE MATCHING ALGORITHM

This section introduces the proposed **FSMDTW** (**F**ast **S**ubsequence **M**atching algorithm for **D**ynamic **T**ime **W**arping) approach. **FSMDTW** discards most of the candidates through LB_{M+}, resulting in a significant improvement over previous work. To further improve the efficiency of subsequence matching problems, we need to optimize the existing DTW lower bound function from the aspects of runtime efficiency or lower bound effect. Therefore, FSMDTW also includes two important optimizations for the existing DTW lower bound. The first one is to improve the effectiveness of the LB_{Keogh} by reprocessing query Q. Moreover, the implementation of $LB_{Petitjean}$ is also improved by computing the projection vector and its envelope on the fly, ultimately improving the efficiency of FSMDTW.

4.1 Overview of FSMDTW

The Algorithm 5 describes the pseudocode of **FSMDTW**. **FSMDTW** firstly divides series $S \in \mathbb{R}^n$ into segments and then solves the problem for each segment separately. To avoid missing subsequences located at the junction between two segments, these segments must overlap each other by a length of $m-1$. As shown in Figure 5, the series S is cut into $p = \lceil \frac{n}{\ell - m + 1} \rceil$ segments, which are denoted as $\{T^{(i)}\}_{i=1}^{p}$.

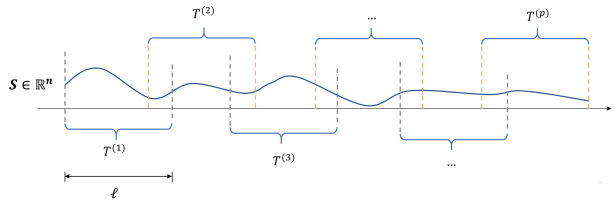


Figure 5: FSMDTW divides S into overlapping segments of length ℓ, where $\frac{\ell}{m} \geq 2$ and $\ell = 2^i$ for some $i \in \mathbb{N}^*$.

The length of each segment is set to ℓ, where $\ell \geq 2m$ to ensure that the average cost of the proposed lower bound is still $O(\log m)$. Besides, the value of ℓ is set to some power of 2 such that the Fast Fourier Transform can efficiently perform the divide-and-conquer strategy on $T^{(i)}$.

After that, **FSMDTW** solve subsequence matching on each ℓ-length segment by checking $\ell - m + 1$ subsequences of length m. **FSMDTW** avoids calculating DTW by cascading utilization of lower bounds. Specifically, lower bounds with cheaper overhead are first computed, and candidates whose lower bound is greater than threshold ϵ are discarded. **FSMDTW** firstly computes LB_{M+} with Algorithm 2 and Algorithm 3 Line (7-9). If the DTW lower bound is greater than ϵ, the candidate can be safely discarded, thereby not necessary to compute the consequent lower bounds and DTW. In fact, the proposed LB_{M+} pruned about 98% of the candidates in our experiments.

Then, the surviving candidates are further checked using lower bounds whose time complexity are $O(m)$ in Lines (14-17). Based on the characteristics of the subsequence matching problem, we designed a tighter lower bound than LB_{Keogh}, called LB_{KE}, which will be discussed in Section 4.2. In addition, we further compute the $LB_{Petitjean}$ and improve its efficiency by computing the elements on the fly in Section 4.3.

Finally, only if a candidate passes all the tests of above lower bounds, the DTW distance will be calculated in the last step, and the existing result set will be updated.

Algorithm 5: $FSMDTW(S, Q, \epsilon)$

Input: Time series $S \in \mathbb{R}^n$, query $Q \in \mathbb{R}^m$ and threshold $\epsilon > 0$.
Output: $\{S_{(i)} | DTW(\hat{S}_i, \hat{Q}) \leq \epsilon\}$.

1 $Q \leftarrow \hat{Q}$; $\mathbb{A} \leftarrow \emptyset$; // normalize Q if necessary
2 $\ell \leftarrow \{2^i\}$, where $4m < \ell \leq 8m$ and $i \in \{1, 2, \ldots, \lceil \log_2(n) \rceil\}$;
3 $p \leftarrow \lceil \frac{n}{\ell - m + 1} \rceil$; $q \leftarrow \ell - m + 1$;
4 $DT \leftarrow$ ***MakeBin***(Q) ; // see Section 4.2
5 **foreach** $t \in \{1, 2, ..., p\}$ **do**
6 $T^{(t)} \leftarrow (s_{tq+1}, s_{tq+2}, \ldots, s_{tq+\ell})$; // the tth segment
7 $\{lb_i\}_{i=1}^{q} \leftarrow$ ***LB_Q***$(Q, T^{(t)})$; // Algorithm 2
8 **if** $|\{i | lb_i \leq \epsilon\}| \geq \log \ell$ **then**
9 $\{lbT_i\}_{i=1}^{q} \leftarrow$ ***LB_T***$(Q, T^{(t)})$; // Algorithm 3
10 **foreach** $1 \leq i \leq q$ **do**
11 $lb_i \leftarrow \max\{lb_i, lbT_i\}$;
12 **foreach** $i \in \{1, 2, ..., q\}$ **do**
13 **if** $\epsilon \geq lb_i$ **then**
14 **if** $\epsilon \geq LB_{KE}(\hat{S}_{i+tq-1}, Q)$ **then**
15 $lb_{Keogh} \leftarrow LB_{Keogh}(Q, \hat{S}_{i+tq-1})$;
16 **if** $\epsilon \geq lb_{Keogh}$ **then**
17 $\Delta \leftarrow LB_{Petitjean}(Q, \hat{S}_{i+tq-1})$;
18 **if** $\epsilon \geq \Delta$ *AND* $\epsilon \geq DTW(Q, \hat{S}_{i+tq-1})$ **then**
19 $\mathbb{A} \leftarrow \mathbb{A} \cup \{S_{i+tq-1}\}$;
20 **return** $\mathbb{A}$;

4.2 Enhancing LB_{Keogh}

LB_{KE} is a tighter lower bound than LB_{Keogh}, and can be efficiently approximated in subsequence matching. Given series $X = (x_1, x_2, \ldots, x_m) \in \mathbb{R}^m$ and $Q = (q_1, q_2, \ldots, q_m) \in \mathbb{R}^m$, the procedure to compute $LB_{KE}(X, Q)$ is described as Algorithm 6. Firstly, we discrete the range (Q_{min}, Q_{max}) into B equal-length intervals $\{I_1, I_2, \ldots, I_B\}$ with $H(v) = \lfloor \frac{B(v - Q_{min})}{Q_{max} - Q_{min}} + 1 \rfloor$, where Q_{min} and Q_{max} are the minimum and maximum values of Q, respectively. Then, a distance look-up table $DT \in \mathbb{R}^{B * m}$ is defined as,

$$DT[k][i] = \min_{i-w \leq j \leq i+w, t \in I_k} \left\{ (t - q_j)^2 \right\}, \tag{26}$$

where $DT[k][i]$ stores the minimum distance between q_j and any x_i that satisfies $H(x_i) = k$.

Clearly, the value of DT is only related to Q. Since Q remains constant throughout the subsequence matching problem, DT can be shared by all candidates and the time cost of establishing DT is negligible. The process of calculating LB_{KE} is shown in Algorithm 6. Lines (3-7) of Algorithm 6 is exactly computing $LB_{Keogh}(X, Q)$. The main difference lies in the utilization of the look-up table DT to

enhance the effectiveness of the lower bound. Therefore, in practice LB_{KE} is always "tighter" than LB_{Keogh}.

Algorithm 6: $LB_{KE}(X, Q)$

Input: $X = (x_1, \ldots, x_m) \in \mathbb{R}^m$, $Q = (q_1, q_2, \ldots, q_m) \in \mathbb{R}^m$
Output: **lower bound of** $LB_{KE}(X, Q)$.

```
dist ← LB_KimFL(X, Q)^2 ;
// DT can be shared if Q does not change
Compute DT ∈ R^{B*m} with Equation (26) ;
foreach 4 ≤ i ≤ m − 3 do
    if x_i ≥ U_i^Q then
        dist ← dist + (x_i − U_i^Q)^2 ;
    else if x_i ≤ L_i^Q then
        dist ← dist + (x_i − L_i^Q)^2 ;
    else
        dist ← dist + DT[H(x_i)][i] ;
return √dist;
```

4.3 Accelerating $LB_{Petitjean}$

$LB_{Petitjean}$ is the tightest known DTW lower bound of $O(m)$ time complexity, but it is much slower than LB_{Keogh}. Therefore, $LB_{Petitjean}$ is the last DTW lower bound to check in FSMDTW. $LB_{Petitjean}$ is defined based on LB_{Keogh} and LB_{kimFL}, which have been calculated in **FSMDTW** previously. Therefore, we only need to compute the remaining part of $LB_{Petitjean}(X, Q)$.

The original implementation of $LB_{Petitjean}$ abandons the calculation immediately once the cumulative lower bound exceeds ϵ [31]. However, it still needs to fully compute the projection vector P and its upper and lower envelopes $\mathbb{U}^P$ and $\mathbb{L}^P$ for each candidate subsequence.

This paper improves the efficiency of $LB_{Petitjean}$ according to the following observations: Firstly, the projection vector P and its envelopes ($\mathbb{L}^P$ and $\mathbb{U}^P$) are utilized element by element in $LB_{Petitjean}$. Furthermore, P, $\mathbb{L}^P$, and $\mathbb{U}^P$ can be calculated element by element without incurring additional overhead. Therefore, we compute $LB_{Petitjean}$ efficiently by computing P, $\mathbb{U}^P$ and $\mathbb{L}^P$ element by element on demand, which avoids computing most envelopes, thereby significantly speeding up $LB_{Petitjean}$.

5 EXPERIMENTS

5.1 Experiment Setup

Hardware. All experiments are carried out on a server with 64GB RAM, an Intel CORE i5-12500 CPU and 3TB SSD storage.

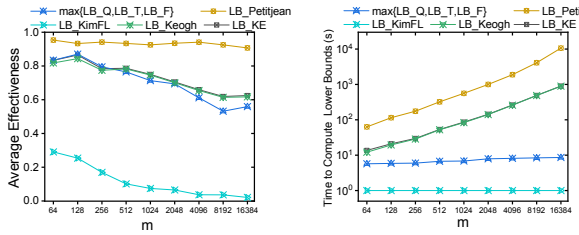
Algorithms The state-of-the-art index-free algorithm are UCR-Suite [23] and its variant UCR-MON Suite[4]. We obtained the C/C++ code for these algorithms from their official website [5]. UCR-Suite was originally designed to search for only the top-1 candidate for a given query. We have slightly modified it to adapt to the ϵ ranged subsequence matching problem. Our algorithm, *FSMDTW*, is also implemented in C/C++ and available online [6].

Datasets. This paper employs both synthetic and real-world datasets, which are described in Table 3.

5.2 The Tightness and Efficiency of the Proposed Lower Bound

This subsection evaluates LB_{M+} against the other DTW lower bounds in terms of both efficiency and effectiveness. The effectiveness of DTW lower bounds is measured by *Tightness of Lower Bound* (TLB) [12], which is defined as the average ratio between the estimated lower bound and the exact DTW value. A higher TLB value indicates that the lower bound is more effective, as it allows for the elimination of a greater number of candidates.

5.2.1 Effectiveness of lower bounds. Figure 6(a) reports the average effectiveness on the five datasets mentioned in Section 5 of different DTW lower bounds as m changes. LB_Q and LB_T represent the lower bounds computed by Algorithm 2 and Algorithm 3, respectively. The effectiveness of the proposed lower bound is measured as $max\{LB\_Q, LB\_T, LB\_F\}$, where LB_F is a special case of LB_{M+} where all elements of the mask vector is set to 1.



(a) Effectiveness vs. m (b) Efficiency vs. m

Figure 6: The effectiveness (measured by TLB) and efficiency (normalized by setting the cost of LB_{KimFL} as 1) of DTW lower bounds with respect to query length m, without considering the impact of early stopping techniques.

The experimental results indicate that the effectiveness of LB_{KimFL} decreases rapidly with increasing query length m, while the effectiveness of other lower bounds decrease slowly. This rapid decline in effectiveness of LB_{KimFL} is attributed to its consideration of only six pairs of elements, which hinders its ability to reflect the dissimilarity between longer series. In contrast, all other lower bounds are less affected by the query length m because they potentially utilize every element of the series in their computation.

5.2.2 Efficiency of lower bounds. Figure 6 (b) illustrates the efficiency of different DTW lower bounds as m changes. Experiments indicate that the time overhead of these DTW lower bounds are consistent with their theoretical time complexity. As m increases, we observe that 1) the cost of LB_{Keogh} and $LB_{Petitjean}$ increases

[5] https://www.cs.ucr.edu/~eamonn/UCRsuite.html and https://github.com/MonashTS/UCR-Monash

[6] https://github.com/QW1k0YA/A-Fast-Index-free-Subsequence-Matching-Algorithm-for-Dynamic-Time-Warping

Table 3: Datasets Used in the Experiments

Dataset	Length	Description of Dataset
RW	1×10^9	An artificial dataset where the difference of adjacent points follows the standard normal distribution.
ECG [7]	1.7×10^9	An electrocardiograms dataset collected from 15 humans with a sampling frequency of 1KHz.
EEG [8]	2.4×10^9	An electroencephalogram dataset collected from 22 humans, sampled at a frequency of 256Hz.
DNA [9]	1.7×10^{10}	A DNA dataset consists of nucleotide sequences derived from various species.
TEMP [10]	1.9×10^9	A dataset contains monthly average temperature data for China from January 1901 to December 2023.

linearly with m; 2) the cost of LB_{KimFL} remains stable; 3) the overhead of $max\{LB\_T, LB\_Q, LB\_F\}$ increases very slowly, reflecting its $O(\log m)$ time complexity.

Although LB_{M+} is less effective than LB_{Keogh}, it significantly outperforms in terms of efficiency. For example, when $m = 1024$, the proposed method is nearly 20 times faster than LB_{Keogh}. When $m = 16384$, our method is more than two orders of magnitude faster than LB_{Keogh}, and the speed advantage continues to increase with the length of the sequence. In addition, LB_{KE} is always tighter than LB_{Keogh}. The calculation of LB_{KE} is slightly slower than that of LB_{Keogh}, but the difference is almost negligible.

5.3 The Efficiency of FSMDTW on Short and Medium Length Queries

Now, we compare the proposed *FSMDTW* algorithm with UCR-Suite in terms of efficiency. The reasonable distance threshold ϵ varies with the length of the query and the datasets. Therefore, we determine the threshold ϵ based on selectivity, which is the ratio between the size of result and the total number of potential candidates. For example, if the selectivity of the query is set to 10^{-8}, then ϵ is adjusted to ensure that the number of results is $10^{-8} \times (n - m + 1)$.

The selectivity of the queries is set to 10^{-9}, 10^{-8}, 10^{-7} and 10^{-6}. The limit of the warping path, w, is set to $\lceil 0.01m \rceil$, $\lceil 0.02m \rceil$, and $\lceil 0.05m \rceil$ respectively. Five query series are selected for each combination of the parameters. Figure 7 reports the average runtime of the subsequence matching algorithms in different datasets.

The results show that FSMDTW consistently outperforms UCR-Suite and UCR-MON Suite in all datasets. Across the five datasets included in the experiments, the proposed approach demonstrates an average speedup of 2.01 $\sim$ 3.22 times compared to the UCR-MON Suite and is 3.32 $\sim$ 4.24 times faster than the UCR-Suite. This advantage can be attributed to the lower bound proposed in this paper. It discards around 98% of the candidates on average in the experiments, thus significantly reducing the time cost in the computation process of the lower bound function compared to existing works.

In addition, the experimental results show that our advantage over UCR-Suite and UCR-MON Suite increases with increasing query length m. The reason is that, as the increase of m, the speed advantage of the fast lower bound method proposed in this paper over the existing DTW lower bounds continuously increases. Therefore, the advantage of FSMDTW is more significant for longer queries. Additionally, the space complexity of FSMDTW is $O(\ell \log \ell)$, where ℓ is the length of the sequence segment (recalling Algorithm 5). In all experiments presented in this paper, the maximum memory usage never exceeds 20MB. Therefore, we do not report the detailed space usage of the Algorithms.

5.4 The Scalability for Longer Queries

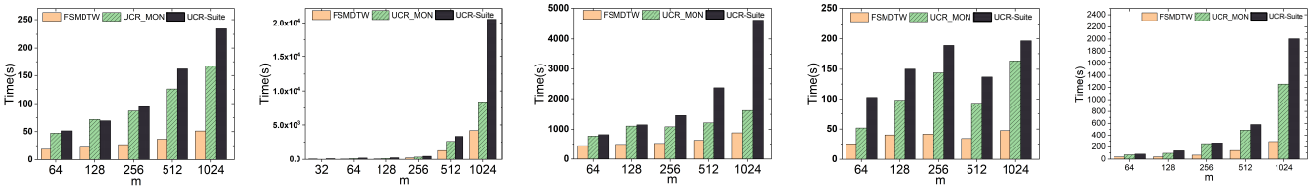
This subsection evaluates the performance of the subsequence matching algorithms against longer query series. The selectivity of queries is set to 10^{-8}. Table 4 reports the execution time of the algorithms on different datasets.

Table 4: Time taken to perform subsequence matching (s)

		Dataset				
m	Algorithm	RW	EEG	DNA	ECG	TEMP
2^{11}	UCR	329.5	10514.3	1537.8	68.5	178.3
	UCR-MON	201.1	4153.9	1234.5	40.7	113.1
	FSMDTW	17.5	1664.4	415.2	27.4	67.8
2^{12}	UCR	802.3	5492.9	1659.7	97.8	355.6
	UCR-MON	505.2	4250.6	1034.4	62.6	231.4
	FSMDTW	24.8	2665.9	313.1	29.6	66.1
2^{13}	UCR	1370.1	72448.6	2859.1	80.3	384.3
	UCR-MON	812.5	43507.5	1728.1	51.6	252.4
	FSMDTW	20.3	21403.3	371.1	30.2	35.8
2^{14}	UCR	3506.5	190570.7	12724.5	232.5	337.6
	UCR-MON	2079.2	87014.9	8750.8	133.9	214.1
	FSMDTW	30.4	42806.5	379.5	36.5	44.4

The experimental results show that the proposed *FSMDTW* is significantly faster than the other competitors on longer queries. The average speedup is 20.4 when query length reaches 2^{14}. This significant advantage can be attributed to the proposed DTW lower bounds, which only cost $O(\log m)$ time. Our algorithm performs better on the random walk dataset than on the other datasets. The reason is likely to be that the random walk dataset is more consistent with the assumptions we made about the distribution during the calculation of mask vectors. Therefore, LB_{M+} filters out most of the candidates on this dataset, which results in FSMDTW achieving better performance.

[7] https://physionet.org/content/butqdb/1.0.0/
[8] https://physionet.org/content/chbmit/1.0.0
[9] https://hgdownload.cse.ucsc.edu/goldenpath/hg38/bigZips/,the DNA sequence is converted into a numerical time series by assigning values to nucleotides: 'A' (+2), 'G' (+1), 'C' (-1), and 'T' (-2).
[10] https://www.tpdc.ac.cn/zh-hans/data/71ab4677-b66c-4fd1-a004-b2a541c4d5bf



(a) RW (b) EEG (c) DNA (d) ECG (e) TEMP

Figure 7: Average time taken by subsequence matching algorithms on different datasets.

Furthermore, it can be observed that the time overhead does not necessarily increase monotonically with the query length m. This is because a significant portion of the time is spent calculating the DTW lower bounds. Consequently, the filtering efficiency of candidates plays a crucial role in the overhead of subsequence matching. As the length of the query sequence increases, variations in filtering efficiency may lead to a decrease in the overall time consumption of the algorithm.

5.5 Influence of Selecting Mask Vectors

This section investigates the impact of mask vector selection strategy on the effectiveness of lower bounds, and the result is shown in Figure 8. We tested three different strategies as the baseline. The random strategy generates a mask vector by setting each element of the mask vector M to follow an independent binomial distribution with the same probability to be either 1 or 0. The all-one strategy simply configures the mask vector with all elements set to 1. The heuristic strategy refers to the strategy we used to set the mask vectors. Finally, "heuristic + all one" represents the maximum of lower bounds generated with the mask vector generated by all one and the heuristic strategy.

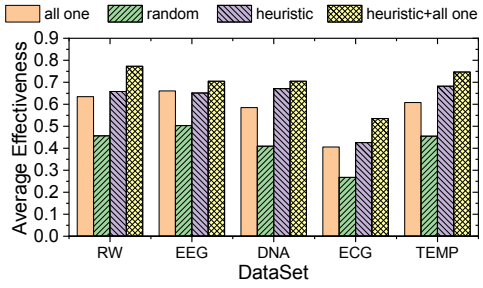


Figure 8: The effectiveness of LB_{M+} under different strategies.

The results show that the random strategy is essentially always the worst, as it does not leverage any knowledge and blindly masks elements compared to the all-one vector. In most cases, mask vectors based on heuristic methods outperform the all-one mask vector because they selectively ignore elements that may worsen the result. Finally, the method that combines the all-one mask vector with the heuristic mask vector has the best lower bound effectiveness. This demonstrates the rationality of the approach adopted in this paper, which involves using different mask vectors multiple times to compute LB_{M+}.

5.6 Ablation Study

We conducted ablation experiments on the three main optimizations proposed in this paper, namely, the optimized algorithms for LB_{M+}, LB_{KE}, and the acceleration of $LB_{Petitjean}$. The results are shown in Table 5. We set $w = \lceil 0.05m \rceil$ and $m = 256$ and report the execution time of the algorithm after removing or replacing certain modules. The experimental results show that the improvements proposed in this paper contribute to the performance of the proposed *FSMDTW* Algorithm. However, the degree of performance improvement varies across different datasets, which may be attributed to data distribution differences.

Table 5: Ablation Study on Removing Different Modules

	Dataset				
Runtime (s)	RW	EEG	DNA	ECG	TEMP
Remove LB_{M+}	63.1	652.6	680.7	100.9	503.0
Replace LB_{KE} with LB_{Keogh}	34.6	339.9	516.2	94.4	210.9
Remove acceleration of $LB_{Petitjean}$	25.7	332.6	516.2	82.3	189.2
FSMDTW	24.0	225.6	514.9	82.3	182.5

6 CONCLUSION

The subsequence matching problem based on DTW plays a key role in time series analysis. This paper designs the first algorithm capable of generating DTW lower bounds in average $\Theta(\log m)$ time, demonstrating a substantial improvement in efficiency over the existing approaches. By meticulously integrating this fast lower bound, enhancing the pruning power of LB_{Keogh} and accelerating the computation of $LB_{Petitjean}$, this paper proposes the FSMDTW algorithm for the subsequence matching problem. Experimental results demonstrate that FSMDTW significantly outperforms the state-of-the-art algorithm, particularly for long queries, with an improvement of up to an order of magnitude.

ACKNOWLEDGMENTS

This work is supported by National Natural Science Foundation of China (NSFC) (62232005, 62202126). We also appreciate the support from China Mobile Group Heilongjiang Co. Ltd. on our research, the research is jointly completed by both parties.

REFERENCES

[1] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. 2015. Tight hardness results for LCS and other sequence similarity measures. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*. IEEE, 59–78.

[2] Sara Alaee, Kaveh Kamgar, and Eamonn Keogh. 2020. Matrix profile XXII: exact discovery of time series motifs under DTW. In *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE, 900–905.

[3] Sara Alaee, Ryan Mercer, Kaveh Kamgar, and Eamonn Keogh. 2021. Time series motifs discovery under DTW allows more robust discovery of conserved structure. *Data Mining and Knowledge Discovery* 35 (2021), 863–910.

[4] Sam Anzaroot and Andrew McCallum. 2013. *UMass Citation Field Extraction Dataset*. Retrieved May 27, 2019 from http://www.iesl.cs.umass.edu/data/data-umasscitationfield

[5] Ardalan, Aarabi, , , Kamran, Kazemi, , , Reinhard, Grebe, , , and Hamid. 2009. Detection of EEG transients in neonates and older children using a system based on dynamic time-warping template matching and spatial dipole clustering - ScienceDirect. *NeuroImage* 48, 1 (2009), 50–62.

[6] Zemin Chao, Hong Gao, Yinan An, and Jianzhong Li. 2022. The inherent time complexity and an efficient algorithm for subsequence matching problem. *Proceedings of the VLDB Endowment* 15, 7 (2022), 1453–1465.

[7] Zemin Chao, Hong Gao, Dongjing Miao, Jianzhong Li, and Hongzhi Wang. 2025. An Amortized O(1) Lower Bound for Dynamic Time Warping in Motif Discovery. *IEEE Transactions on Knowledge and Data Engineering* 37, 5 (2025), 2239–2252. https://doi.org/10.1109/TKDE.2025.3544751

[8] Nikan Chavoshi, Hossein Hamooni, and Abdullah Mueen. 2016. Debot: Twitter bot detection via warped correlation.. In *Icdm*, Vol. 18. 28–65.

[9] Matthieu Herrmann and Geoffrey I Webb. 2021. Early abandoning and pruning for elastic distances including dynamic time warping. *Data Mining and Knowledge Discovery* 35, 6 (2021), 2577–2601.

[10] J. Jing, J. Dauwels, T. Rakthanmanon, E. Keogh, and M. B. Westover. 2016. Rapid annotation of interictal epileptiform discharges via template matching under Dynamic Time Warping. *Journal of Neuroscience Methods* 274 (2016), 179–190.

[11] Rong Kang, Chen Wang, Peng Wang, Yuting Ding, and Jianmin Wang. 2018. Matching Consecutive Subpatterns over Streaming Time Series. In *Web and Big Data*, Yi Cai, Yoshiharu Ishikawa, and Jianliang Xu (Eds.). Springer International Publishing, Cham, 90–105.

[12] Eamonn Keogh, Kaushik Chakrabarti, Michael Pazzani, and Sharad Mehrotra. 2001. Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and information Systems* 3 (2001), 263–286.

[13] Eamonn Keogh and Chotirat Ann Ratanamahatana. 2005. Exact indexing of dynamic time warping. *Knowledge and information systems* 7 (2005), 358–386.

[14] Sang-Wook Kim, Sanghyun Park, and Wesley W Chu. 2001. An index-based approach for similarity search supporting time warping in large sequence databases. In *Proceedings 17th international conference on data engineering*. IEEE, 607–614.

[15] Daniel Lemire. 2009. Faster retrieval with a two-pass dynamic-time-warping lower bound. *Pattern Recognition* 42, 9 (2009), 2169–2180. https://doi.org/10.1016/j.patcog.2008.11.030

[16] Daniel Lemire. 2009. Faster retrieval with a two-pass dynamic-time-warping lower bound. *Pattern recognition* 42, 9 (2009), 2169–2180.

[17] Michele Linardi and Themis Palpanas. 2018. Scalable, variable-length similarity search in data series: The ULISSE approach. *Proceedings of the VLDB Endowment* 11, 13 (2018), 2236–2248.

[18] Frank Madrid, Shima Imani, Ryan Mercer, Zachary Zimmerman, Nader Shakibay, and Eamonn Keogh. 2019. Matrix profile xx: Finding and visualizing time series motifs of all lengths using the matrix profile. In *2019 IEEE International Conference on Big Knowledge (ICBK)*. IEEE, 175–182.

[19] Ricards Marcinkevics, Steven Kelk, Carlo Galuzzi, and Berthold Stegemann. 2019. Discovery of Important Subsequences in Electrocardiogram Beats Using the Nearest Neighbour Algorithm. (2019).

[20] Victor Maus, Gilberto Câmara, Ricardo Cartaxo, Alber Sanchez, Fernando M Ramos, and Gilberto R De Queiroz. 2016. A time-weighted dynamic time warping method for land-use and land-cover mapping. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 9, 8 (2016), 3729–3739.

[21] Abdullah Mueen, Eamonn Keogh, Qiang Zhu, Sydney Cash, and Brandon Westover. 2009. Exact discovery of time series motifs. In *Proceedings of the 2009 SIAM international conference on data mining*. SIAM, 473–484.

[22] Vit Niennattrakul, Dechawut Wanichsan, and Chotirat Ann Ratanamahatana. 2010. Accurate subsequence matching on data stream under time warping distance. In *New Frontiers in Applied Data Mining: PAKDD 2009 International Workshops, Bangkok, Thailand, April 27-30, 2009. Revised Selected Papers 13*. Springer, 156–167.

[23] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. 2012. 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*. 262–270.

[24] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. 2013. Addressing big data time series: Mining trillions of time series subsequences under dynamic time warping. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 7, 3 (2013), 1–31.

[25] Chotirat Ann Ratanamahatana and Eamonn Keogh. 2005. Three myths about dynamic time warping data mining. In *Proceedings of the 2005 SIAM international conference on data mining*. SIAM, 506–510.

[26] Hiroaki Sakoe and Seibi Chiba. 1978. Dynamic programming algorithm optimization for spoken word recognition. *IEEE transactions on acoustics, speech, and signal processing* 26, 1 (1978), 43–49.

[27] Diego F Silva, Rafael Giusti, Eamonn Keogh, and Gustavo EAPA Batista. 2018. Speeding up similarity search under dynamic time warping by pruning unpromising alignments. *Data Mining and Knowledge Discovery* 32 (2018), 988–1016.

[28] Johannes Stübinger and Lucas Schneider. 2020. Epidemiology of coronavirus COVID-19: Forecasting the future incidence in different countries. In *Healthcare*, Vol. 8. MDPI, 99.

[29] Chang Wei Tan, François Petitjean, and Geoffrey I Webb. 2019. Elastic bands across the path: A new framework and method to lower bound DTW. In *Proceedings of the 2019 SIAM International Conference on Data Mining*. SIAM, 522–530.

[30] Charles Van Loan. 1992. *Computational frameworks for the fast Fourier transform*. SIAM.

[31] Geoffrey I. Webb and François Petitjean. 2021. Tight lower bounds for dynamic time warping. *Pattern Recognition* 115 (2021), 107895. https://doi.org/10.1016/j.patcog.2021.107895

[32] Jiaye Wu, Peng Wang, Ningting Pan, Chen Wang, Wei Wang, and Jianmin Wang. 2019. Kv-match: A subsequence matching approach supporting normalization and time warping. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 866–877.

[33] Chin-Chia Michael Yeh, Nickolas Kavantzas, and Eamonn Keogh. 2017. Matrix profile VI: Meaningful multidimensional motif discovery. In *2017 IEEE international conference on data mining (ICDM)*. IEEE, 565–574.

[34] Chin-Chia Michael Yeh, Yan Zhu, Liudmila Ulanova, Nurjahan Begum, Yifei Ding, Hoang Anh Dau, Diego Furtado Silva, Abdullah Mueen, and Eamonn Keogh. 2016. Matrix profile I: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets. In *2016 IEEE 16th international conference on data mining (ICDM)*. Ieee, 1317–1322.

[35] Sheng Zhong and Abdullah Mueen. 2024. MASS: distance profile of a query over a time series. *Data Mining and Knowledge Discovery* (2024), 1–27.

[36] Yunyue Zhu and Dennis Shasha. 2003. Warping indexes with envelope transforms for query by humming. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. 181–192.

[37] Yan Zhu, Zachary Zimmerman, Nader Shakibay Senobari, Chin-Chia Michael Yeh, Gareth Funning, Abdullah Mueen, Philip Brisk, and Eamonn Keogh. 2016. Matrix profile ii: Exploiting a novel algorithm and gpus to break the one hundred million barrier for time series motifs and joins. In *2016 IEEE 16th international conference on data mining (ICDM)*. IEEE, 739–748.