CIVET: Exploring Compact Index for Variable-Length Subsequence Matching on Time Series

Haoran Xiong
Fudan University
hrxiong20@fudan.edu.cn

Hang Zhang
Fudan University
zhanghang21@m.fudan.edu.cn

Zeyu Wang
Fudan University
zeyuwang21@m.fudan.edu.cn

Zhenying He[*]
Fudan University
zhenying@fudan.edu.cn

Peng Wang[*]
Fudan University
pengwang5@fudan.edu.cn

X. Sean Wang
Fudan University
xywangcs@fudan.edu.cn

ABSTRACT

Nowadays the demands for managing and analyzing substantially increasing collections of time series are becoming more challenging. Subsequence matching, as a core subroutine in time series analysis, has drawn significant research attention. Most of the previous works only focus on matching the subsequences with equal length to the query. However, many scenarios require support for efficient variable-length subsequence matching. In this paper, we propose a new representation, Uniform Piecewise Aggregate Approximation (UPAA) with the capability of aligning features for variable-length time series while remaining the lower bounding property. Based on UPAA, we present a compact index structure by grouping adjacent subsequences and similar subsequences respectively. Moreover, we propose an index pruning algorithm and a data filtering strategy to efficiently support variable-length subsequence matching without false dismissals. The experiments conducted on both real and synthetic datasets demonstrate that our approach achieves considerably better efficiency, scalability, and effectiveness than existing approaches.

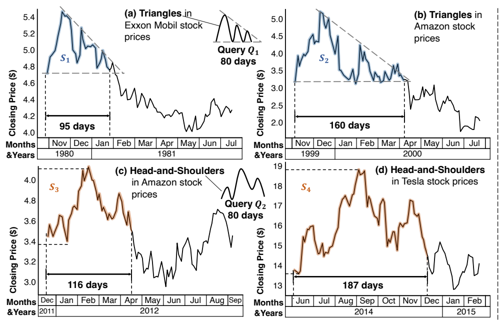
PVLDB Reference Format:
Haoran Xiong, Hang Zhang, Zeyu Wang, Zhenying He, Peng Wang, and X. Sean Wang. CIVET: Exploring Compact Index for Variable-Length Subsequence Matching on Time Series. PVLDB, 17(9): 2123-2135, 2024.
doi:10.14778/3665844.3665845

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/CIVET-TS/CIVET.

1 INTRODUCTION

Time series has become prevalent due to numerous applications generating extensive collections of time-stamped data [13, 15, 42], necessitating advanced analysis techniques for valuable insights. As a core subroutine of time series analysis, subsequence matching has attracted significant attention and research effort [25, 30, 31, 39].

Informally, subsequence matching finds subsequences from a long sequence that are similar to a given query sequence. Many subsequence matching approaches focus on matching the subsequences with the same length as the query [8, 29]. However, it has been demonstrated that many applications call for sequence matching approaches that allow for variable length [12, 18]. We illustrate it with an example below.



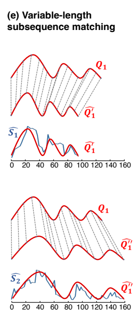


Figure 1: Variable-length Subsequence Matching on four Stock datasets. The Triangles pattern (Q_1) searches datasets (a) and (b), yielding results S_1 and S_2. The Head-and-shoulders pattern (Q_2) searches datasets (c) and (d), yielding results S_3 and S_4. (e) shows the process of variable-length subsequence matching between Q_1 and S_1, S_2.

Example 1. In stock trading, technical analysis is a financial method to identify investigation opportunities by studying historical trading activity [1]. As a concrete example, Figure 1 presents four sequences of stock prices, which conform to two classic chart patterns in technical analysis, *triangles* (S_1, S_2 in Figure 1(a) and (b)) and *head-and-shoulders* (S_3, S_4 in Figure 1(c) and (d)) [9], depicted with highlighted lines. Q_1 and Q_2 are utilized as query sequences for two chart patterns, employed for subsequence matching. □

Different subsequences though adhering to the same pattern, may have different lengths. We name this phenomenon as *global scaling*. In this case, only finding subsequences of the same length as query will lose many meaningful results. To solve this problem, uniform scaling has been proposed to align variable-length subsequences [18, 31]. With uniform scaling, the query sequence is uniformly stretched or shrunk to different lengths and aligned with the target subsequence, thus solving global scaling in the

[*]Corresponding authors

Proceedings of the VLDB Endowment, Vol. 17, No. 9 ISSN 2150-8097.
doi:10.14778/3665844.3665845

time dimension. This technique is widely employed in actual application [10, 18, 33, 41]. In addition, to focus on the shape of subsequences and eliminate the effect offset shifting and amplitude, z-normalization [17, 30] is often used before distance calculation, which still works in the context of uniform scaling.

In this paper, we focus on the variable-length subsequence matching problem under uniform scaling and z-normalization. During the variable-length subsequence matching, the query sequence is first uniformly-scaled to variable lengths and then find similar subsequences of the corresponding lengths [12]. Figure 1 shows two concrete examples. Before calculating the distances, the query Q_1 is scaled to sequences $Q_1^{'}$ and $Q_1^{''}$ whose lengths are the same as S_1 and S_2 and then computing distance (say, Euclidean Distance) between them after z-normalization. The hat symbols (e.g., $\hat{Q_1^{'}}$) indicate z-normalized sequences.

Compared with fixed-length subsequence matching, the variable-length subsequence matching problem is more challenging. First, the variable-length case has a much larger search space. Given a length-l query subsequence, fixed-length matching only needs to verify all *equi-length* subsequences in the database, whereas variable-length matching needs all the subsequences in the concerned range of lengths. Second, calculating distance between subsequences of different lengths is more time-consuming and it involves a large number of redundant calculations when computing distances between query and different subsequences of long time series. Therefore, it is essential to design an efficient index and query algorithm to avoid unnecessary computations.

Numerous works have studied variable-length *sequence* matching problem under uniform scaling [10, 12, 18, 41]. However, these approaches do not consider the subsequences of the time series in the database, despite that the length of the subsequences are in the user's concerned range. Thus, the search space of this problem is substantially smaller than ours. Other works [31, 33] study the variable-length *subsequence* matching problem. Nonetheless, these methods cannot take effect with z-normalization due to the design of distance bounds. Thus, their results are prone to get disturbed by the offset shifting and amplitude of subsequences.

In this paper, we propose an extended PAA that can eliminate the influence of global scaling between time series. Besides, we have meticulously designed a compact index structure, which stores similar but variable-length subsequences closely, possessing the ability to handle scaled queries of different lengths at once. Moreover, our methodologies in representation and index structures have been meticulously designed. They can efficiently facilitate exact top-K queries without incurring any false dismissals combined with the respective lower bound distances. In addition, we formulate enveloping sequences by utilizing the monotonicity of the mean and standard deviation values in z-normalization. Leveraging these enveloping sequences, we introduce a robust lower bounding distance designed to efficaciously eliminate redundant distance computations in the context of uniform scaling and normalization.

In summary, facilitated by a new representation technique, we design a compact index structure supporting efficient variable-length subsequence matching with the help of index pruning and data filtering strategies. We call the approach as CIVET (Compact Index for Variable-length subsequencE matching on Time series). CIVET is experimentally proved to be more advanced than the SOTA approaches in terms of approximate matching accuracy and exact matching performance under uniform scaling and normalization.

Table 1: Table of Symbols

Symbols	Description
$T = (t_1, t_2, ..., t_n)$	Time series
$\|S\|$	Length of S
$T_{i,l}$	Subsequence of T (from i to $i + l - 1$)
$\widehat{S}$	Z-normalized S
Q^p	Q scaled to length p with uniform scaling
S_i	The i-th segment of S in UniSeg
r_i	The last point index of i-th segment
$D_{ed}(\cdot, \cdot)$	Euclidean distance
$D_{dtw}(\cdot, \cdot)$	DTW distance
$D_{\ddot{u}sn}(\cdot)$	Uniform scaling distance with normalization

The contributions of this paper can be summarized as follows:

- The Uniform Piecewise Aggregate Approximation (UPAA) is introduced to manage variable lengths, aligning feature representations while retaining essential properties of PAA, thus enhancing robustness against global scaling.
- A new indexing method is designed, constructing a compact index structure by grouping adjacent subsequences and subsequently grouping subsequences with similar features.
- Leveraging lower bounding properties, we propose effective index pruning and data filtering techniques, both tailored for global scaling and z-normalization and compatible with ED and DTW distances.

The paper is organized as follows: Section 2 presents the formal problem statement and method overview. Section 3 extends the PAA to represent variable-length time series. Sections 4 and 5 introduce the details of index construction and matching algorithms. Section 6 presents and discusses the experimental results. Section 7 discusses related works. Section 8 concludes the paper.

2 PROBLEM STATEMENT

2.1 Preliminaries and Problem Formulation

Time series is a sequence of values listed in time order, denoted as $T = (t_1, t_2, \cdots, t_n)$, where $n = |T|$ is the length of T. Subsequence $T_{i,l}$ of time series T is a length-l contiguous sequence within T, which starts from position i. Formally, we denoted $T_{i,l}$ as $T_{i,l} = (t_i, t_{i+1}, \cdots, t_{i+l-1})$, where $1 \leq i \leq |T| - l + 1$. Later in this paper, we refer to query sequence and subsequence as Q and S for distinction. T is used to refer to a long sequence specifically. For any subsequences $S = (s_1, s_2, \cdots, s_n)$, we use μ^S and σ^S to denote the *mean value* and *standard deviation* of S respectively.

Definition 1 (Euclidean Distance (ED)). *Given Q and S with the same length l, ED between them is $D_{ed}(Q, S) = \sqrt{\sum_{i=1}^{l} (q_i - s_i)^2}$.*

ED computes the distance between two sequences with one-to-one map, while DTW eliminates local misalignment with one-to-many map. The warping path is used to describe the mapping relation between two sequences.

Definition 2 (Warping Path). *Given two length-l sequences, Q and S, a warping path is denoted as $A = (a_1, a_2, \ldots, a_{|A|})$. The x-th element $a_x = (i, j)$ is a pair of values representing the mapping between Q_i and S_j. We use $a_x.fst$ and $a_x.snd$ to refer to the first and second values of a_x. A warping path satisfies the following constraints: (1) $1 \leq i, j \leq l$, $a_1 = (1, 1)$, and $a_{|A|} = (l, l)$, (2) $0 \leq a_{x+1}.fst - a_x.fst \leq 1$ and $0 \leq a_{x+1}.snd - a_x.snd \leq 1$.*

Definition 3 (Constrained Dynamic Time Warping Distance). *Given two length-l sequences Q and S, and the time warping constraint c, the constrained dynamic time warping distance between them is defined as, $D_{dtw}(Q, S) = \arg\min_A \sqrt{\sum_i^{|A|} (q_{a_i.fst} - s_{a_i.snd})^2}$. According to the Sakoe-Chiba constraint [32], any element a_x in the warping path A satisfies that $|a_x.fst - a_x.snd| \leq c$.*

Definition 4 (Uniform Scaling). *Given a length-n sequence Q and a length p, the uniform scaling stretches up (if $n < p$) or shrinks down (if $n > p$) S to a length-p time series $Q^p = (q_1^p, q_2^p, \cdots, q_p^p)$, where*

$$q_i^p = q_{\lceil i * \frac{n}{p} \rceil}, 1 \leq i \leq p$$

Definition 5 (Z-Normalized Series). *Given a length-n time series S, a normalized series of S, is defined as, $\hat{S} = (\hat{s}_1, \hat{s}_2, \cdots, \hat{s}_n)$, where $\hat{s}_i = \frac{s_i - \mu^S}{\sigma^S}$, $1 \leq i \leq n$.*

Under the influence of time series length, ED between shorter time series is more likely small, even though they could be less similar. Therefore, the length norm is adopted to eliminate the influence of different lengths. It divides the distance by $\sqrt{l}$, where l is the length of the sequences [26].

Definition 6 (Uniform Scaling Distance with Norm). *Given Q and S, the uniform scaling distance with both z-norm and length-norm between them is defined as follows:*

$$D_{usn}(Q, S) = \frac{D(\widehat{Q^l}, \hat{S})}{\sqrt{l}}, l = |S|.$$

Definition 6 scales Q to length $|S|$, while it is also possible to scale S to $|Q|$. Both approaches possess similar capabilities in eliminating global scaling. Here we stay consistent with the preceding works [31]. Distance D in D_{usn} can be either D_{ed} or D_{dtw} depending on the concrete scenario. We denote them as D_{usn}^{ed} and D_{usn}^{dtw}. In this paper, when saying two sequences are similar or have similar patterns, we mean that they have a small distance under D_{usn}. That is, when scaling to the same length, the two sequences have a very small ED/DTW.

Problem 1 (Top-K Subsequence Matching within D_{usn}). *Given a time series T, a length range $[l_{min}, l_{max}]$ and an integer K, for any Q, the top-K matching is to find a set of subsequences $\mathbb{R} = \{S_1, S_2, \cdots, S_K\} \subseteq \mathbb{A}$, where $\mathbb{A}$ contains all subsequences of T whose lengths satisfy the length range, such that, $\forall S \in \mathbb{R}$ and $\forall S' \in \mathbb{A} - \mathbb{R}$, $D_{usn}(Q, S) \leq D_{usn}(Q, S')$.*

2.2 iSAX Index Family

Our work preserves the main structure of iSAX index, thus we briefly review the related techniques in this part.

2.2.1 Representation. The representation technique summarizes time series into a lower-dimensional representation to estimate the approximate distance between them efficiently. We review the PAA used by iSAX index [16, 40].

The Piecewise Aggregate Approximation (PAA) [16, 40] splits a sequence S into disjoint equal-length segments and represents each segment with the mean of its values, which transforms S into a m-dimensional representation $PAA(S)$, where $m = \lfloor \frac{|S|}{length\ of\ segment} \rfloor$.

Referring to the proposition [11, 16], PAA gives a lower bounding distance for the Euclidean distance between two sequences.

Proposition 1 (PAA Lower Bound). *Given two time series Q and S such that $|Q| = |S|$, we have*

$$D_{ed}(Q, S) \geq \sqrt{l_{seg} \cdot \sum_{i=1}^{m} (PAA(Q)_i - PAA(S)_i)^2}. \tag{1}$$

where l_{seg} is the length of segment, $m = \lfloor |S|/l_{seg} \rfloor$.

The SAX [24] and iSAX [34] representations are also adopted in iSAX index to to reduce storage space and facilitate index construction. Briefly described, they discretize each coefficient of $PAA(S)$ as a binary string, referred to as $SAX(S)$ and $iSAX(S)$.

2.2.2 Index Structure. When given a predefined segment length, iSAX index [34, 38] follows the same behavior to represent all sequences as PAAs and construct a tree-like index structure on top of them. The index consists of three types of nodes (root node, inner node, and leaf node). The root node has at most 2^m child nodes, while the inner node has only 2 child nodes. Each node has a distinct iSAX representation representing all the sequences in its subtree. Combining the Proposition 1, iSAX index can guarantee no false dismissals when pruning tree nodes during the matching procedure. The index also supports incremental data insertion and dynamic update of tree structure [2].

2.3 Approach Overview

Our work systematically tackles this problem from three interconnected aspects: data representation, indexing, and query processing. The framework and basic design approach are illustrated in Figure 2.

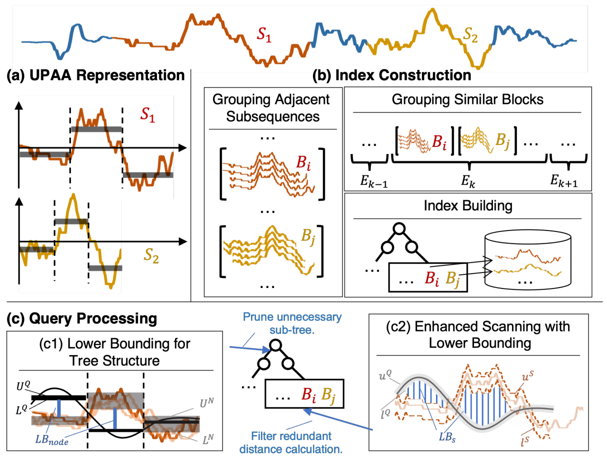


Figure 2: CIVET Framework

UPAA Representation. As shown in Figure 2(a), we propose Uniform Piecewise Aggregate Approximation (UPAA), which effectively summarizes sequences of different lengths and provides an exact lower bound of the uniform scaling distance with norm between two sequences for pruning.

Index Construction. We design a compact iSAX-based index to manage massive subsequences based on UPAA, where *adjacent* subsequences are organized into blocks (e.g., B_i, B_j in Figure 2(b)), and *similar* blocks are organized into envelops (e.g., E_k). In this way, subsequences are compactly stored in our index and can be efficiently accessed based on similarity.

Query Processing. We further design exact and approximate subsequence querying algorithms that can efficiently prune the subtrees based on the lower bound provided by UPAA when traversing the index (see Figure 2(c1)), and can also filter out unpromising subsequences when scanning the subsequences inside a node without any false dismissals (see Figure 2(c2)).

3 EXTENDING PAA FOR GLOBAL SCALING

We extend PAA as UPAA to handle the global scaling among sequences and present the lower bound properties of UPAA.

3.1 Uniform PAA

Since PAA splits both query and database subsequences into equal-length segments, it cannot solve the global scaling phenomenon. As shown in Figure 3(a), although S_1 and S_2 exhibit the same pattern, the equi-length segmentation of PAA fails to capture this similarity.

To tackle this problem, we adopt a new segmentation strategy. Instead of fixing the length of each segment, we fix the total number of segments. Formally, we first define the segmentation method and then extend the PAA with this method.

Definition 7 (Uniform Segmentation (UniSeg)). *Given a sequence $S = (s_1, s_2, \cdots, s_n)$ and the number of segments m, S is segmented as m parts, denoted as $UniSeg(S) = (S_1, S_2, \cdots, S_m)$. The i-th segment is defined as $S_i = (s_{r_{i-1}+1}, \cdots, s_{r_i})$, where $r_i = \lfloor \frac{i \times n}{m} \rfloor$, for $1 \le i \le m$, and initially, $r_0 = 0$.*

Definition 8 (Uniform PAA (UPAA)). *Given a sequence $S = (s_1, s_2, \cdots, s_n)$ and the number of segments m, we compress and represent $UniSeg(S)$ as a m-dimension vector, denoted as $UPAA(S) = (\mu_1, \mu_2, \cdots, \mu_k)$, where μ_i is the mean value of S_i. The i-th coefficient $UPAA(S)_i$ is denoted as $\mu_i(S)$ interchangeably.*

Given a dataset, m is the same for database and query subsequences, despite different lengths. In this way, for similar sequences with global scaling, the corresponding segments after UniSeg will tend to be similar. As demonstrated in Figure 3(b), UPAA (with $m = 3$) effectively transforms S_1 and S_2 into similar and same-dimensional representations.

Similar to PAA, UPAA also possesses the lower bound property.

Theorem 1 (UPAA Lower Bound). *Given two time series Q and S such that $|Q| = |S|$, the number of segments m, we have that,*

$$D_{ed}(Q, S) \ge \sqrt{\left\lfloor \frac{|S|}{m} \right\rfloor \cdot \sum_{i=1}^{m} (UPAA(Q)_i - UPAA(S)_i)^2}. \quad (2)$$

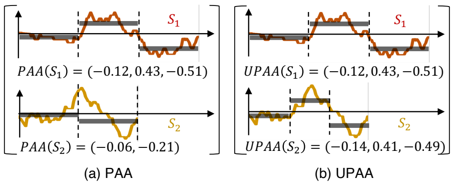


Figure 3: PAA and UPAA. S_1 and S_2 have a small D_{usn} (a) but PAA summarize them with values of large differences. (b) UPAA summarize them with closer values.

Proof. According to the definition of UniSeg, a given sequence might be split into segments of different lengths. But the difference among the lengths of all segments does not exceed one. Let $l_{seg} = \lfloor |S|/m \rfloor$, we have that,

$$r_i - r_{i-1} = l_{seg} \text{ or } l_{seg} + 1, \text{for } 1 \le i \le m. \quad (3)$$

According to the corollary in [40], for the i-th segment, we have,

$$(r_i - r_{i-1}) \cdot (\mu_i(Q) - \mu_i(S))^2 \le \sum_{j=r_{i-1}+1}^{r_i} (q_j - s_j)^2 \quad (4)$$

Now, we can easily prove the correctness by scaling the polynomial coefficient as follows,

$$\begin{aligned} D_{ed}(Q, S) &\ge \sqrt{\sum_{i=1}^{m} (r_i - r_{i-1}) \cdot (\mu_i(Q) - \mu_i(S))^2} \\ &\ge \sqrt{\left\lfloor \frac{|S|}{m} \right\rfloor \cdot \sum_{i=1}^{m} (\mu_i(Q) - \mu_i(S))^2}. \end{aligned} \quad (5)$$

□

Note that when the sequence length is divisible by m, the formula in Theorem 1 and the formula in Proposition 1 have the same meaning. Consequently, UPAA enhances PAA with capabilities to represent and align variable-length sequences without losing the original properties of PAA.

3.2 Lower Bound for a Set of Time Series

Usually, we need to estimate the distance between a query sequence and a set of variable-length subsequences. So, we infer the lower bounding distance for this situation. First, we scale the query to all possible lengths and use two vectors to delimit minimal and maximal UPAA representations of the scaled query sequences. Formally, for a query sequence Q and a set of sequences $\mathbb{S}$, and the number of segments m, we denote the minimal and maximal UPAAs as L^Q and U^Q, respectively, such that for $1 \le i \le m$.

$$L_i^Q = \min\left(\left\{\mu_i(Q^{|S|})\right\}\right), U_i^Q = \max\left(\left\{\mu_i(Q^{|S|})\right\}\right), \forall S \in \mathbb{S}, \quad (6)$$

where $Q^{|S|}$ means scaling the query Q to the length of sequence S using uniform scaling.

Similarly, for the sequences in $\mathbb{S}$, we can use two vectors, $L^{\mathbb{S}}$ and $U^{\mathbb{S}}$, to enclose the minimal and maximal UPAA coefficients of all the sequences in the set $\mathbb{S}$,

$$L_i^{\mathbb{S}} = \min(\mu_i(S)), U_i^{\mathbb{S}} = \max(\mu_i(S)), \forall S \in \mathbb{S}. \quad (7)$$

Now we have the lower bound for a set of time series.

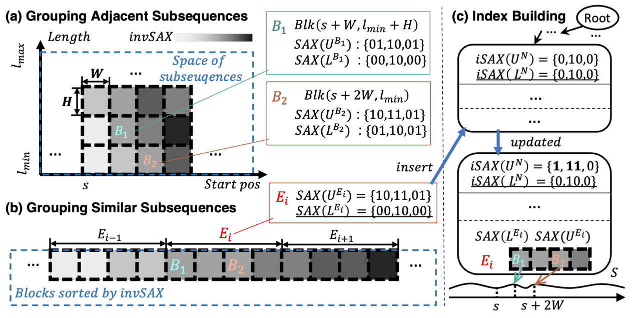

(a) Grouping Adjacent Subsequences
Length
invSAX
l_{max}
l_{min}
W
H
Space of subseuqences
B_1
B_2
s
Start pos
B_1 $Blk(s+W, l_{min}+H)$
$SAX(U^{B_1})$: {01,10,01}
$SAX(L^{B_1})$: {00,10,00}
B_2 $Blk(s+2W, l_{min})$
$SAX(U^{B_2})$: {10,11,01}
$SAX(L^{B_2})$: {01,10,01}
(b) Grouping Similar Subsequences
E_i $SAX(U^{E_i})$ = {10,11,01}
$SAX(L^{E_i})$ = {00,10,00}
E_{i-1}
E_i
E_{i+1}
Blocks sorted by invSAX
insert
(c) Index Building
Root
$iSAX(U^N)$ = {0,10,0}
$iSAX(L^N)$ = {0,10,0}
updated
$iSAX(U^N)$ = {1,11,0}
$iSAX(L^N)$ = {0,10,0}
$SAX(L^{E_i})$ $SAX(U^{E_i})$
S
s
s + 2W

Figure 4: CIVET Index Construction

Theorem 2 (UPAA Lower Bound on Set). *Given a sequence Q and a set of time series $\mathbb{S}$, the number of segments m, we have that,*

$$\min_{S \in \mathbb{S}}\{D_{ed}(Q^{|S|}, S)\} \geq \sqrt{\left\lfloor \frac{l_{min}}{m} \right\rfloor \cdot \sum_{i=1}^{m} \begin{cases} (L_i^{\mathbb{S}} - U_i^Q)^2, if\ L_i^{\mathbb{S}} > U_i^Q \\ (U_i^{\mathbb{S}} - L_i^Q)^2, if\ U_i^{\mathbb{S}} < L_i^Q \\ 0 \qquad\qquad\quad , otherwise \end{cases}} \tag{8}$$

where l_{min} is the minimal length of sequences in $\mathbb{S}$.

Proof. Without loss of generality, we consider a random sequence S in $\mathbb{S}$ here. Before calculating the distance, Q is scaled to the length of S. And it is easy to know that $|S| \geq l_{min}$. So, combining with the Theorem 1, we have that,

$$D_{ed}(Q^{|S|}, S) \geq \sqrt{\left\lfloor \frac{l_{min}}{m} \right\rfloor \cdot \sum_{i=1}^{m} (\mu_i(Q^{|S|}) - \mu_i(S))^2}. \tag{9}$$

According to the definition of L and U for the query and the set of sequences, we easily prove the correctness of Equation 8. □

Therefore, UPAA enables the iSAX family index to obtain the capability to process variable-length data effectively while retaining the ability of index pruning without false dismissals.

4 INDEX CONSTRUCTION

In this section, we present our compact index. We provide two techniques named block summarization (Section 4.1) and envelope summarization (Section 4.2) to compact the redundant information of subsequence. Then, we describe the procedure of building CIVET index (Section 4.3).

4.1 Grouping Adjacent Subsequences

In this part, we provide a representation method to summarize sets of overlapping subsequences succinctly.

A specific subsequence is determined by its start position and length. Considering subsequences as points on a two-dimensional coordinate, all the subsequences of a long time series form a two-dimensional space. we depict it as the space of subsequences in Figure 4(a). Thus, subsequences can be divided into small rectangles with width W and height H. The two user-defined parameters, W and H, represent the stepsizes of start position and length, respectively. We refer to the rectangle as Block (Blk). Since there are many overlaps among the subsequences in the same block, their UPAAs tend to be similar. We summarize them with a higher-level representation.

Definition 9 (Block). *Given two parameters W and H, a block Blk(s,l) groups a set of adjacent subsequences in the long sequence S and delimits the UPAA coefficients of these subsequences with two m-dimension vectors, L^B and U^B. The set of subsequences in Blk(s,l) is defined as follows,*

$$\mathbb{S}^B = \{S_{i,l'} | s \leq i < s + W \ and\ l \leq l' < l + H\}. \tag{10}$$

And two vectors, L^B and U^B, satisfy that,

$$L_i^B = min(\mu_i(S')), U_i^B = max(\mu_i(S')), \forall S' \in \mathbb{S}^B. \tag{11}$$

Specific boundary corner cases are not explicitly mentioned in Equation 10 for brevity. But note that the subsequences in the set $\mathbb{S}^B$ must adhere to the sequence length constraints $[l_{min}, l_{max}]$.

Now, we can summarize all subsequences within the space of subsequences using continuous but non-overlapping blocks, as illustrated in Figure 4(a). Each block effectively summarizes the information of the corresponding adjacent sequences using only two low-dimensional vectors and a pointer that references the raw data. We depict two concrete examples of Blocks, B_1 and B_2.

Block summarization allows users to adjust W and H to trade space and time. However, we can only summarize the subsequences with close positions and lengths. The similar subsequences may not be arranged rectangularly in the space of subsequences. Intuitively, in Figure 4, we use similar colors to mark the Blocks with similar representations. For example, the blocks B_1 and B_2 are similar but not adjacent. So, we propose a new method to rearrange and pack up Blocks with similar features to improve the compactness of the index further.

4.2 Grouping Similar Blocks

Block summarizes the information of adjacent subsequences. This part focuses on the blocks with similar UPAA representations but different positions. We call the grouped blocks an Envelope. We refine the sortable representation invSAX [20] for block summarization and then introduce the procedure to group blocks.

4.2.1 InvSAX for block summarization. The invSAX provides the ability to convert the SAX representation of a time series into a sortable representation [20]. Sorted by invSAX, time series with similar SAX will be placed in close positions. So we utilize the invSAX to rearrange blocks in our work.

Given a time series S, $SAX(S)$ represents S with a length-m vector, whose i-th value $SAX_i(S)$ is a binary number. The bits in the binary number indicate the possible value range of $PAA_i(S)$, and the higher bit has more impact on this range. The key idea of invSAX is to sort the time series according to the more important bits, i.e., the higher bits have higher sorting priority. For example, the invSAX representation of SAX(**011**, **101**, **001**) is represented as '**010100111**'.

Here, we refine the invSAX for block summarization. As introduced above, block summarization delimits a block *Blk* with two vectors L^B and U^B. We can merge these two vectors into one named LU^B as follows,

$$LU^B(Blk) = (L_1^B, U_1^B, \cdots, L_m^B, U_m^B), \tag{12}$$

where m is the number of segments.

Now, the invSAX can be easily applied to block summarization straightforwardly. We transform the $LU^B(Blk)$ into SAX representation and then convert the SAX into sortable summarization using the same logic of invSAX. We refer to this sortable summarization as $invSAX^B$ for simplicity. For instance, L^{B_1} and U^{B_1} in Figure 4(a) are respectively given by (**00**, **10**, **00**) and (**01**, **10**, **01**). Consequently, the $invSAX^B$ representation for B_1 would be **001100010001**.

4.2.2 Envelope construction. In this part, we further compact the blocks with similar UPAA representations facilitated by $invSAX^B$.

Given the blocks constructed from a long time series, we append them into an array and sort the array by $invSAX^B$. Thus, the blocks with similar L^B and U^B tend to be placed in close positions of the array. We use an envelope to summarize the blocks in a sliding window.

Definition 10 (Envelope). *Given a length-n array of blocks sorted by $invSAX^B$ $(B_1, B_2, \cdots, B_n)$ and window size ws, we group every ws blocks as an envelope and delimit their UPAAs with two m-dimensional vectors, L^E and U^E. Formally, the set of blocks in the i-th envelope is defined as $\mathbb{S}^{E_i} = \{B_j | (i-1) * ws < j \le min(n, i * ws)\}$. And two vectors L^E and U^E of E_i satisfy that,*

$$L_i^E = min(L_i^B), U_i^E = max(U_i^B), \forall B \in \mathbb{S}^{E_i}. \tag{13}$$

A concrete example is depicted in Figure 4(b). We set the size of the sliding window to 4. To aid in understanding, we provide a more intuitive example in Figure 2(b). When grouping adjacent subsequences, the construction algorithm collects subsequences with similar lengths and starting positions into the same block, as the figure illustrates by B_i and B_j. When grouping similar blocks, B_i and B_j, which have similar UPAA features, are sorted into nearby positions for constructing the envelope.

So far, we compact the subsequences with similar UPAAs in an Env with the help of the summarization methods above. For one envelope, we only need to store two vectors, L^E and U^E, and the pointers of blocks in the envelope. Combining these grouping methods, we can construct a compact and efficient index to support subsequence matching.

4.3 Index Building

Here in this part, we tend to present the procedure to index the envelopes based on iSAX index [34], called CIVET index.

Similar to ULISSE [25], we maintain minimal and maximal iSAX symbols in each node of CIVET index, denoted by $iSAX(L^N)$ and $iSAX(U^N)$. Besides that, in the leaf node, we additionally store the SAX representations of two vectors, L^E and U^E, and pointers to the blocks. CIVET does not contain real data since subsequences overlap extensively.

Before constructing the index structure, we first build the envelopes for subsequences as described in Sections 4.1 and 4.2. Then, we insert these envelopes into CIVET index according to the $SAX(L^E)$ one by one. That is, finding the target leaf node whose $iSAX(L^N)$ contains $SAX(L^E)$. Then we update the nodes in the route from the root node to the leaf node with respect to $iSAX(U^N)$.

Figure 4(c) shows a concrete example of envelope inserting. The E_i is inserted according to the $SAX(L^{E_i})$. Notably, the $iSAX(U^N)$ is also updated to ensure the property of the node representation vectors. Besides, we also depict the pointers of blocks B_1 and B_2.

When building the CIVET index, we adopt the efficient algorithm for block construction [25] and then sort blocks to construct envelopes. However, it will consume a lot of memory to sort all the blocks of long time series. So we utilize the buffer mechanism. We sequentially load part of the raw time series into a fixed-size buffer and then conduct blocks and envelopes in bulk.

Complexity Analysis. For convenience, we let M be the value $l_{max} - l_{min}$, m be the segment number, k be the number of blocks in one envelope and N is the length of long time series. Before utilizing any grouping technique, the magnitude of subsequences is $O(MN)$. while the CIVET shrinks the space complexity to $O(\frac{bMN}{kWH})$, where b means the bytes of each envelope. The time complexity of block construction is $O(\frac{M^2Nm}{W})$. The time complexity of envelope construction is $O(\frac{MN}{WH} lg(\frac{MN}{WH}))$.

5 QUERY PROCESSING

This section provides the matching algorithm, including the lower bounding distance, details on matching algorithms, optimization of scanning to reduce unnecessary distance calculations, and extends these techniques to support DTW.

5.1 Lower Bounding for Envelope and Node

In this part, we propose a lower bounding distance between the query and envelope to prune the candidate envelopes and tree nodes during the exact matching.

First of all, similar to envelope summarization, we use two vectors to summarize the information of a given query. Specifically, We scale the query to all possible lengths in the range $[l_{min}, l_{max}]$, calculate the UPAAs, and delimit them as lower and upper bounds to represent the query. Formally, given m as the number of segment and a query Q, the lower and upper bounds of Q are denoted as L^Q and U^Q respectively, such that,

$$L_i^Q = min(\{\mu_i(\widehat{Q^l})\}),\ U_i^Q = max(\{\mu_i(\widehat{Q^l})\}), \tag{14}$$

where $1 \le i \le m$ and $l \in [l_{min}, l_{max}]$.

The distance between PAA and SAX is provided for the lower bound of the Euclidean distance [34]. Similarly, we refer to the lower and upper breakpoints of SAX value as $\beta_L(\cdot)$ and $\beta_U(\cdot)$. Given a query Q with bounds L^Q and U^Q and an envelope E with bounds L^E and U^E, we define a lower bounding distance between them as,

$$LB_{env}(Q, E) = \tag{15}$$

$$\sqrt{\frac{\eta}{m}} \sqrt{\sum_{i=1}^{m} \begin{cases} (\beta_L(SAX(L^E)_i) - U_i^Q)^2, if\ \beta_L(SAX(L^E)_i) > U_i^Q \\ (\beta_U(SAX(U^E)_i) - L_i^Q)^2, if\ \beta_U(SAX(U^E)_i) < L_i^Q \\ 0 \qquad\qquad , otherwise \end{cases}}$$

As mentioned earlier, UniSeg may produce segments of different lengths. To eliminate the influence of this phenomenon and ensure the correctness of the lower bounding distance, we import the scaling factor η, such that,

$$\eta = \frac{l'}{l'+1}, where\ l' = \lfloor \frac{l}{m} \rfloor \tag{16}$$

l refers to the minimal length among all subsequences in this envelope, and l' indicates the minimal length of segments.

Theorem 3. *Given an envelope E and a query Q, for any subsequence S in the envelope, we have that,*

$$LB_{env}(Q,E) \leq D^{ed}_{usn}(Q,S). \tag{17}$$

Proof. According to the definition of PAA and SAX in [34], we know that $\beta_L(SAX(L^E)_i) \leq L^E_i \leq U^E_i \leq \beta_U(SAX(U^E)_i)$, where β_L and β_U are the lower and higher breakpoint of iSAX [34]. We have,

$$LB_{env}(Q,E) \leq \sqrt{\frac{\eta}{m}}\sqrt{\sum_{i=1}^{m}\begin{cases}(L^E_i - U^Q_i)^2, if\ L^E_i > U^Q_i\\(U^E_i - L^Q_i)^2, if\ U^E_i < L^Q_i\\0 \qquad\qquad , otherwise\end{cases}} \tag{18}$$

Then we consider the scaling factor η. According to the monotonicity of the function $f(x) = 1/(x+1)$, combining with $0 < l_{min} < |S|$, we have,

$$\eta = \frac{l'}{l'+1} = \frac{\left\lfloor\frac{l_{min}}{m}\right\rfloor}{\left\lfloor\frac{l_{min}}{m}\right\rfloor + 1} \leq \frac{\left\lfloor\frac{|S|}{m}\right\rfloor}{\left\lfloor\frac{|S|}{m}\right\rfloor + 1}. \tag{19}$$

Besides, it is easy to know,

$$|S|/m \leq \lfloor |S|/m \rfloor + 1. \tag{20}$$

From Equation 19 and Equation 20 we infer that,

$$\frac{\eta}{m} \leq \frac{\left\lfloor\frac{|S|}{m}\right\rfloor}{|S|} \tag{21}$$

Now, according to Equation 8, Equation 18 can be derived as,

$$LB_{env}(Q,E) \leq \sqrt{\frac{1}{|S|}}\sqrt{\left\lfloor\frac{|S|}{m}\right\rfloor \cdot \sum_{i=1}^{m}\begin{cases}(L^E_i - U^Q_i)^2, if\ L^E_i > U^Q_i\\(U^E_i - L^Q_i)^2, if\ U^E_i < L^Q_i\\0 \qquad\qquad , otherwise\end{cases}}$$
$$\leq \sqrt{\frac{1}{|S|}} \cdot D_{ed}(\widehat{Q^{|S|}}, \widehat{S}) = D^{ed}_{usn}(Q,S)$$

□

Moreover, we apply the same logic of $LB_{env}(Q,E)$ to lower bound the distance between query Q and node N in CIVET. We only need to replace the envelope's SAX representation with the node's iSAX representation. Formally, given a query Q with bounds L^Q and U^Q and a node N with bounds L^N and U^N, we define a lower bounding distance between them as,

$$LB_{node}(Q,N) = \tag{22}$$
$$\sqrt{\frac{\eta'}{m}}\sqrt{\sum_{i=1}^{m}\begin{cases}(\beta_L(iSAX(L^N)_i) - U^Q_i)^2, if\ \beta_L(iSAX(L^N)_i) > U^Q_i\\(\beta_U(iSAX(U^N)_i) - L^Q_i)^2, if\ \beta_U(iSAX(U^N)_i) < L^Q_i\\0 \qquad\qquad , otherwise\end{cases}}$$

Here, the η' is set as $\frac{l'}{l'+1}$, where $l' = \lfloor\frac{l_{min}}{m}\rfloor$. Similar to LB_{env}, LB_{node} also retains lower-bound properties. Its logic is akin to that of Theorem 3, and thus will not be elaborated here. We depict the detailed calculation of LB_{node} in Figure 2(c1) for an intuitive illustration. The black bar and the gray area, respectively, represent the minimal and maximal UPAAs of scaled queries (L^Q, U^Q in Equation 6) and subsequences in the node (L^N, U^N). The blue bar represents the numerical calculations of LB_{node} in Equation 22.

5.2 Search Algorithm

We utilize the lower bounding distances in the previous subsection to prune unnecessary sub-trees and indicate the visiting order of tree nodes, which supports an efficient approximate search. Then, we refine the results with an exact search procedure.

Algorithm 1 matches the top-K nearest neighbors of query Q and returns the distance of K-th nearest neighbor, $KThBsf$, and the top-K results, R^K. Firstly, we initialize variables. R^K is a max heap used to record the current top-K optimal results, and the variable $KThBsf$ records the largest distance in R^K. The *heap* accepts a node and a LB_{node} distance. The heap orders the inserted nodes in descending order of the lower bounding distances. Initially, we insert the root node into the heap with a zero distance (Line 1-3).In the main loop, we first get the closest node from the heap (Line 5). If the LB_{node} of the node is greater than or equal to the $KThBsf$, the R^K is already the exact result of the top-K search (Line 6-7). We also adopt early-stopping logic for approximate matching procedure, controlling the total number of visiting leaf nodes (Line 8-9). If the node is terminal, we iterate all the envelopes and check the lower bounding distance between E and Q. The envelope is skipped directly if the lower bounding distance equals or exceeds the $KThBsf$. Otherwise, we calculate the exact results using *checkEnv*. Here, the *checkEnv* calculates distances for every subsequence in the envelope and updates the top-K results in $KThBsf$ and R^K (Line 10-13). Later in

Algorithm 1: *searchAlgorithm*

```
   Data: $K, Q, USI, maxVisit$.
   Result: $KThBsf, R^K$.
1  $KThBsf \leftarrow \infty$, Initialize $R^K$; // Max-heap with capability K
2  Initialize heap; // Min-heap
3  $heap.add(USI.root, 0)$;
   // Getting approximate results
4  while heap is not empty do
5      $n \leftarrow heap.pop()$;
6      if $n.dist >= KThBsf$ then
7          return $KThBsf, R^K$; // Got the exact top-K results
8      if Number of visited leaves >= $maxVisit$ then
9          break;
10     if $n.node$ is leaf node then
11         for $E$ in $n.node.envs$ do
12             if $LBenv(Q, E) < KThBsf$ then
13                 $KThBsf, R^K \leftarrow checkEnv(Q, E, KThBsf, R^K)$;
14     else if $n.node$ is internal or root node then
15         for $childNode$ in $n.node.children$ do
16             $heap.add(childNode, LBnode(Q, childNode))$;
   // Getting exact results
17 for $E$ in $USI.sequentialEnvs$ do
18     if $LBenv(Q, E) < KThBsf$ then
19         $KThBsf, R^K \leftarrow checkEnv(Q, E, KThBsf, R^K)$;
20 return $KThBsf, R^K$;
```

Section 5.3, Algorithm 2 will enhance this brute-force procedure with an effective filtering strategy. If the node is an internal or root node, we insert its child nodes into the heap with the lower bounding distances LB_{node} (Line 14-16).

Till now, if the algorithm does not obtain the exact results, we adopt the sequential checking procedure to refine the final results. We maintain sorted envelopes, which allows us to scan the raw data only once to get the final results. The algorithm sequentially processes the envelopes. If an envelope can not be filtered by LB_{env}, we calculate the exact distance between the query and subsequences in the envelope (Line 17-19).

5.3 Enhanced Scanning with Lower Bounding

In Algorithms 1, if an envelope cannot be filtered by LB_{env}, we must calculate all the distances between query and subsequences in the envelopes. In this part, we propose a new lower bounding distance to accelerate the distance calculation.

Lower Bounding. Subsequences at the same position but with varying lengths yield subtle differences after z-normalization, which results in a large amount of redundant calculation. By exploiting the monotonicity in z-normalization, we construct enveloping sequences to delimit the maximum and minimum values of normalized subsequences. Then, we propose a new lower-bound distance LB_s based on these enveloping sequences to skip unnecessary distance calculation.

Given a query Q and a block $B(s, l)$, and parameters W and H. When calculating the distances between the query and the subsequences in this block, query only has to be scaled into lengths within the range $[l, l + H - 1]$ rather than all the possible lengths. Therefore, we can delimit a tighter lower and upper bound for query, denoted as l^Q and u^Q, respectively, such that,

$$\begin{aligned} l_i^Q &= min(\{\widehat{Q_i^{l'}} \mid l' \in [l, l+H-1]\}), \\ u_i^Q &= max(\{\widehat{Q_i^{l'}} \mid l' \in [l, l+H-1]\}) \end{aligned} \tag{23}$$

where $1 \le i \le l$.

Now, we consider all the subsequences with the same start position s' in the block B. So we have $s' \in [s, s + W - 1]$, and the length of these subsequences $l' \in [l, l + H - 1]$. We enclose these subsequences with two enveloping sequences, l^S and u^S. For brevity, we use S to denote $T_{s',l'}$. Here, we show the bounds as follows,

$$l_i^S = \begin{cases} \dfrac{S_i - \mu_{max}}{\sigma_{max}}, if\ S_i > \mu_{max} \\ \dfrac{S_i - \mu_{max}}{\sigma_{min}}, if\ S_i \le \mu_{max} \end{cases} \quad u_i^S = \begin{cases} \dfrac{S_i - \mu_{min}}{\sigma_{min}}, if\ S_i > \mu_{min} \\ \dfrac{S_i - \mu_{min}}{\sigma_{max}}, if\ S_i \le \mu_{min} \end{cases} \tag{24}$$

where $1 \le i \le l$.

Till now, we build the lower bound $LB_s(Q, B, s')$ between query and subsequences starting at the same position s' in the block B,

$$LB_s(Q, B, s') = \sqrt{\frac{1}{l+H-1}} \sqrt{\sum_{i=1}^{l} \begin{cases} (l_i^S - u_i^Q)^2, if\ l_i^S > u_i^Q \\ (u_i^S - l_i^Q)^2, if\ u_i^S < l_i^Q \\ 0, otherwise \end{cases}}. \tag{25}$$

Theorem 4. *Having a block $B(s, l)$ and a query Q, for any subsequence S starting at the position s' in this block B, the following inequality is satisfied,*

$$LB_s(Q, B, s') \le D_{usn}^{ed}(Q, S). \tag{26}$$

Proof. First, we prove the correctness of bounds l^S and u^S, using the monotonicity of μ and σ in the definition of Z-normalization. Without loss of generality, we consider $\sigma > 0$. Let $f(x) = \frac{x-\mu}{\sigma}$. By taking the partial derivative of $f(x)$, we have

$$\frac{\partial f(x)}{\partial \mu} = -\frac{1}{\sigma}, \ \frac{\partial f(x)}{\partial \sigma} = -\frac{x-\mu}{\sigma^2}.$$

For the lower bound l_i^S, we first consider the case of $S_i > \mu_{max}$. Thus, we have $\frac{\partial f(x)}{\partial \sigma} < 0$. To obtain the lower bound, we set σ to the maximal value σ_{max}. For μ, we have $\frac{\partial f(x)}{\partial \mu} < 0$. So, μ needs to take the maximal value μ_{max}. Therefore, in the case of $S_i - \mu_{max} > 0$, $L_i^{s'}$ takes the value of $\frac{S_i - \mu_{max}}{\sigma_{max}}$. Similarly, we can prove the case of $S_i \le \mu_{max}$ and the upper bound u_i^S.

Thus, considering any subsequence $S = (s_1, s_2, \cdots, s_{|S|})$, starting at s' in the block B, we have $l_i^S \le \hat{s}_i \le u_i^S$. It is easy to prove that the $LB_s(Q, B, s')$ is a lower bounding distance for $D_{usn}^{ed}(Q, S)$. □

Figure 2(c2) presents an illustration of LB_s. The grey stripes represent the bounds calculated by Equation 23. The red dashed lines indicate the upper and lower bounds in Equation 24, and the blue shaded area represents the schematic of the LB_s calculation.

Searching Algorithm. Now, the procedure of searching an envelope can be accelerated using LB_s, presented in Algorithm 2. For each block contained by an envelope (Line 1), we iterate every start position in the block (Line 2-3). Before calculating the exact distances, we check whether to skip the calculation using LB_s (Line 4). The calculation of concrete distance is conducted for each subsequence starting at the specific position (Line 5-6). We adopt the online normalization technique [30] when computing the LB_s. Now, we can replace the function *checkEnv* in Algorithms 1 with *checkEnvEnhanced* in Algorithm 2 for faster searching.

Algorithm 2: *checkEnvEnhanced*

```
Data: $Q$, $E$, $KThBsf$, $R^K$.
Result: $KThBsf$, $R^K$.
1 for $B$ in $E.blocks$ do
2     $s \leftarrow B.startPos$;
3     for $i \leftarrow 0$ to $W - 1$ do
4         if $LBs(Q, B, s + i) < bsf$ then
5             for $S$ starting at $s + i$ in the block $B$ do
6                 Calculate $Dusn(Q, S)$, update $KThBsf$ and $R^K$;
7 return $KThBsf$, $R^K$;
```

Complexity analysis. We use l_{avg} to represent the average length of subsequences in a block. The time complexity of calculating LB_s is $O(l_{avg})$ for each start position. Using α as the pruning ratio of LB_s, the average time complexity of *checkEnvEnhanced* is $O\,(wsW(l + (1 - \alpha)Hl))$, while *checkEnv* consumes $O(wsWHl)$ time. Therefore, when the pruning ratio of LB_s is high, some precomputation can help us save a amount of real distance calculation.

5.4 Supporting DTW Distance

To handle the local misalignment between scaled queries and target subsequences, adopting the concept of LB_{Keogh} [19] (which constructs upper and lower-bound sequences for the query sequence to incorporate variations in time axis), we construct a boundary envelope for the query to enable UPAA tolerant the temporal misalignment and support the cDTW distance.

Given a length-n query sequence Q and the time warping constraint c, according to [19], the enveloping sequences for cDTW distance are constructed as,

$$\begin{aligned} u_i^{keogh}(Q) &= max(Q_{max(1,i-c)}, \cdots, Q_{min(i+c,n)}), \\ l_i^{keogh}(Q) &= min(Q_{max(1,i-c)}, \cdots, Q_{min(i+c,n)}) \end{aligned} \tag{27}$$

, where $1 \leq i \leq n$. These two sequences u^{keogh} and l^{keogh} form a length-n envelope to enclose the original sequence Q, which helps to calculate the lower bounding distance for cDTW to accelerate the query processing.

Now we reformulate the lower and upper bounds for query in Section 5.1 and Section 5.3 with a similar idea of [19].

First, we reconstruct the lower and upper bounds in Equation 14, denoted as L^{Qdtw} and U^{Qdtw}, such that,

$$\begin{aligned} L_i^{Qdtw} &= min(\{\mu_i(l^{keogh}(\widehat{Q^l})) \mid l \in [l_{min}, l_{max}]\}), \\ U_i^{Qdtw} &= max(\{\mu_i(u^{keogh}(\widehat{Q^l})) \mid l \in [l_{min}, l_{max}]\}). \end{aligned} \tag{28}$$

Similarly, we restate the lower and upper enveloping sequences in Equation 23, denoted as l^{Qdtw} and u^{Qdtw}, satisfying that,

$$\begin{aligned} l_i^{Qdtw} &= min(\{l^{keogh}(\widehat{Q_i^{l'}}) \mid l' \in [l, l+H-1]\}), \\ u_i^{Qdtw} &= max(\{u^{keogh}(\widehat{Q_i^{l'}}) \mid l' \in [l, l+H-1]\}). \end{aligned} \tag{29}$$

By replacing the enveloping sequences in LB_{env} and LB_s with l^{Qdtw} and u^{Qdtw}, we obtain the new LB_{env}^{dtw} and LB_s^{dtw} suitable for D_{usn}^{dtw}. Referring to the property of enveloping sequences [19], we can easily prove the correctness of these lower bounding distances similar to the Theorem 3 and Theorem 4. Therefore, the CIVET index and query algorithms can be adapted to the cDTW distance.

5.5 Discussion

CIVET is designed for finding similar subsequences with the closest D_{usn}. Users can use a pattern as the query sequence and find subsequences with the same pattern. CIVET does not support constrained matching and non-normalized matching, as well as ad-hoc semantic search. Nonetheless, CIVET can be extended to many real-world applications. For example, one can use CIVET to find subsequences containing multiple patterns by extracting and searching single patterns and then filtering the returned results to obtain the final answer. CIVET can also be extended to support range search by combining the lower bounding distances in Section 5.1 with the user-provided range threshold for timely abandonment.

6 EXPERIMENT

We evaluate the efficiency and effectiveness of CIVET on real and synthetic datasets. All experiments are conducted on a computer running Ubuntu 18.04.6 LTS 64-bit with an Intel(R) Xeon(R) Gold 5215 CPU @ 2.50GHz *multi* 80 CPU, 64GB RAM, and 4TB Dell PERC H730P disk. All methods are implemented with C++.

6.1 Experimental Setup

6.1.1 Datasets. Datasets used in our experiments are listed below.

AGW is a gesture recognition dataset that contains 10 types of gestures acquired by a three-axis accelerometer. **GMA** contains 3D hand trajectories collected with Leap Motion device. **PLAID** contains current and voltage measurements from different appliance types. **GAP** records the active energy consumed from 2006 to 2008 in France [14]. **CAP** contains a periodic EEG activity occurring during NREM sleep phase [36]. **SYN** is a synthetic dataset generated as the sum of a sequence of random steps extracted from a standard Gaussian distribution $\mathcal{N}(0, 1)$.

The AGW, GMA, and PLAID are three small and real datasets containing variable-length sequences provided by UCR Archive [7]. We randomly select sequences for each dataset as queries; the rest are shuffled and concatenated as a long target sequence. Their lengths are 370K, 170K, and 340K, respectively. Length ranges of the query workload are [32,385], [80,360], and [200,684].

For GAP, CAP, and SYN, queries are remolded from random subsequences. We scale subsequences into random lengths within $[l_{min}, l_{max}]$ to generate queries. The additional Gaussian noise is added to the scaled sequences. In the experiments, we set the length range as $[256, 512]$ by default. These three datasets (SYN, GAP, CAP) are large datasets, where the size of each dataset is 10M. Note that the number of candidate subsequences in the long sequence is about 2 billion under these settings.

6.1.2 The Comparison Methods. Three baselines are adopted.

UCR Suite [30] (**UCR** for short) searches the most similar normalized subsequence by scanning the whole time series and speeds up the search processing using some pruning techniques. During the subsequence matching, we scale the given query to every possible length within the length range $[l_{min}, l_{max}]$ and then match top-k subsequences with the UCR Suite.

UCR-US [31] is a representative subsequence matching method that supports normalized distance under uniform scaling. It refines the lower-bound techniques in UCR Suite to suit the scenario of uniform scaling. UCR-US is omitted for comparison under DTW due to lack of support. Besides, UCR and UCR-US can directly execute queries without building indexes.

ULISSE [25], as state of the art in subsequence matching, supports matching the subsequences within the constraint on length range by constructing an iSAX-based index. Similarly, we scale the given query to every possible length and conduct the subsequence matchings with ULISSE. Both CIVET and ULISSE support control the number of visited leaf nodes during the approximate matching procedure. Unless otherwise specified, we set it to 5, the default parameter provided by ULISSE.

UCR and UCR-US are scanning algorithms without an index, while ULISSE and CIVET require constructing indexes. Later, we show that a few queries will amortize the index-building cost.

6.1.3 Parameter Default Setting and Influences. CIVET requires four parameters during the indexing. We provide default values of the parameters that ensure both optimal performance and fair

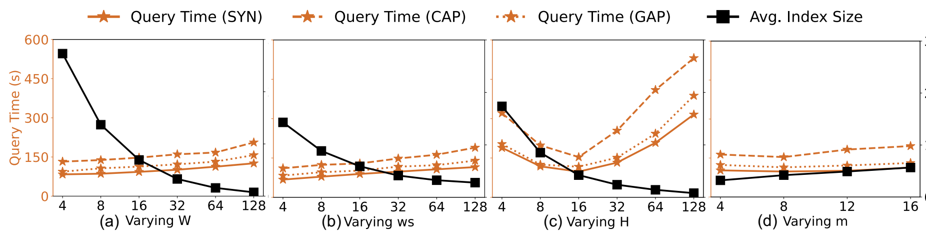


Figure 5: Influence of Parameters

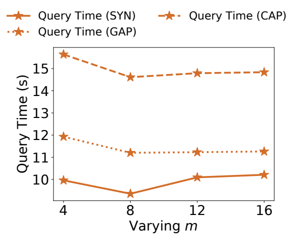


Figure 6: Initial Testing of m

comparison. By default, we set W as $0.1(l_{max} - l_{min})$, H as 16, ws as $\frac{l_{max}-l_{min}}{8}$, m as 8.

We study the influences of parameters on multiple datasets (SYN, CAP, GAP). We adjust one parameter and keep the other as default values to show the changes in query time and index size. Since the structure and size of the index under the same parameter setting are similar, we only show the average size here. The results are depicted in Figure 5.

W and ws, the step size of starting positions and the window size of envelopes. These two parameters are not sensitive to the query time and can fine-tune the time-space tradeoff (see Figure 5 (a) and (b)). Users can decide them according to the space constraint. In our experiments, we use them to control the index size to be nearly equal to the baseline ULISSE in order to ensure a fair comparison.

H, the step size of lengths. The proper H optimizes the effectiveness of both index pruning and scan filtering. In CIVET, there is a sweet point when varying H where the query performance can achieve the optimum. We decide it by pre-experiments. As shown in Figure 5 (c), $H = 16$ is the best point for all three datasets.

m, the number of segments in each subsequence. Experimental results (see Figure 5(d) in the paper) indicate that query efficiency changes slightly when m varies. As m increases, the expressive capability of PAA (and UPAA) improves, but this also leads to higher computational costs [38]. Thus, increasing the number of segments is beneficial up to a point, after which query efficiency starts to decrease [5]. To determine the most effective value of m, we conducted initial testing on a subset of our data. Specifically, we sampled 5% of the subsequences to construct indexes with different m values ($\{4, 8, 12, 16\}$). Then, we executed a series of random queries to assess the efficiency of these indexes. The m value yielding the highest efficiency was selected to construct the complete index. The results of our initial tests, presented in Figure 6, indicate that for our datasets, $m = 8$ offers the best configuration.

The parameter W in CIVET and γ in ULISSE share the same meanings (step size of starting positions). Therefore, we vary them to provide additional details about the index structure, as shown in Table 2. It is evident that, owing to our more compact construction logic, CIVET achieves superior compactness.

6.2 Evaluation of Exact Top-1 Matching

In this part, we have thoroughly explored Exact Top-1 Matching to analyze the efficiency and effectiveness of our method. All experiments in this subsection are conducted on all datasets.

Table 2: Details of Index Structure

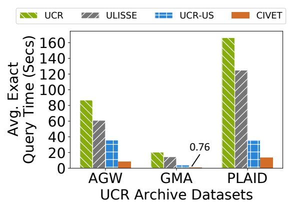
		W for CIVET / γ for ULISSE					
		4	8	16	32	64	128
CIVET	Height	8.7	7.3	6.5	5.4	4.2	3.1
	# Envs	1.3M	620K	314K	156K	78K	39K
	# Leaf Nodes	1843	938	460	227	115	60
	Index Size(MB)	269	138	68	32	16	7.3
ULISSE	Height	9.5	8.6	7.6	6.9	6.1	5.4
	# Envs	2.0M	1.1M	588K	303K	153K	78K
	# Leaf Nodes	3870	2243	1218	724	403	204
	Index Size(MB)	720	361	180	66	33	23

We record the average exact query time and the pruning power of 100 queries. The pruning power refers to the percentage of the total number of subsequences that do not need to calculate concrete distance. Note that ULISSE skips envelopes for efficient matching while UCR-US prunes sets of subsequences with the help of lower bounding distance. CIVET has two steps of pruning procedure. We refer to the index pruning as the first stage (ST1) and the filtering in *checkEnv* as the second stage (ST2).

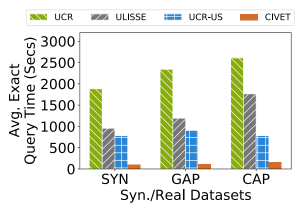
Performance of Exact Top-1 Matching with ED. We first test CIVET and baselines using exact Top-1 matching with ED.

Figure 7(a) and (b) show the average exact Top-1 query time. As depicted, CIVET achieves the acceleration of 2.5x-7.8x compared with UCR-US and 7.3x-11.5x compared with ULISSE on different datasets. Figure 7(c) and (d) report the average pruning power of three methods on these datasets. Due to the grouping of subsequences with similar features during index building, the pruning power of ST1 in our indexing method surpasses that of ULISSE. One distinction is important: unlike ULISSE, we consider all possible lengths at once during the index pruning process. Yet, we can still achieve a higher pruning rate in ST1. This also reflects the capability of UPAA to eliminate the influence of global scaling. In ST2, when setting an appropriate value for H, the algorithm constructs a compact lower-bounding distance for z-normalized subsequences with varying lengths. Therefore, the pruning rate of the scanning algorithm consistently remains very high on different datasets.

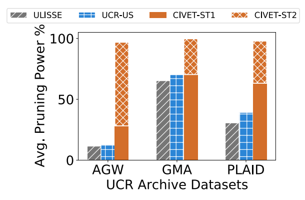
To illustrate the overall time cost of indexing and querying, we show the cumulative time to build the index and answer 20 queries. As shown in Figures 7(e) and 7(f), the index-building overhead brought by CIVET index building is very small. After executing 5 queries, the cumulative time cost of CIVET gets smaller than UCR and UCR-US. Additionally, the results indicate that the Disk I/O time required for CIVET is substantially smaller than ULISSE.



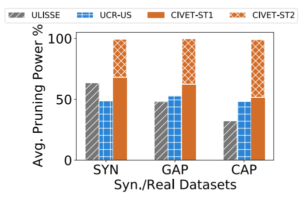
(a) Efficiency on Small Datasets



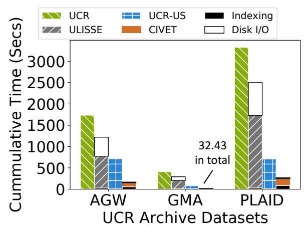
(b) Efficiency on Large Datasets



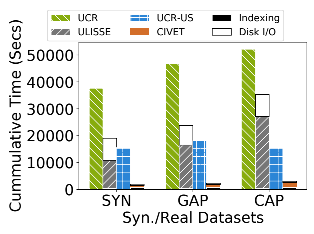
(c) Pruning Power on Small Datasets



(d) Pruning Power on Large Datasets

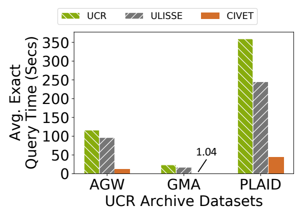


(e) Time Cost on Small Datasets

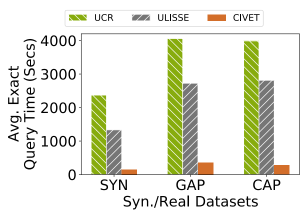


(f) Time Cost on Large Datasets

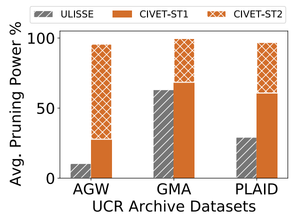
Figure 7: Exact Top-1 Matching with ED



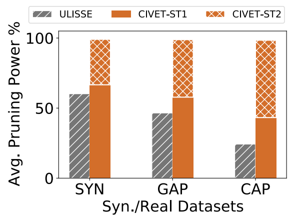
(a) Efficiency on Small Datasets



(b) Efficiency on Large Datasets



(c) Pruning Power on Small Data



(d) Pruning Power on Large Data

Figure 8: Exact Top-1 Matching with DTW

Performance of Exact Top-1 Matching with DTW. CIVET is also extended to support variable-length matching within cDTW distance. So we conduct the exact Top-1 matching with cDTW distance here. Figure 8 shows average query time and pruning effectiveness. As illustrated, CIVET is 5.5x-16.6x faster than ULISSE and 8.1x-22.7x than UCR on different datasets.

In summary, our approach achieves an average of 5x speed-ups than UCR-US, 11x speed-ups than ULISSE, and 15x on UCR on different datasets. The efficiency of matching underscores our method's superiority in variable-length subsequence matching. In addition, the pruning rate and other statistical data also reflect the high effectiveness of UPAA in representing variable-length sequences, and the efficiency and robustness of the query algorithm.

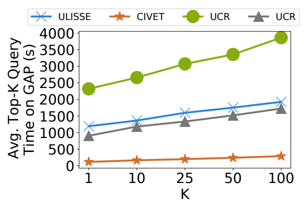
6.3 Exploratory Experiments

To analyze more beneficial aspects of CIVET, we conduct a series of exploratory experiments in this part. All experiments are performed on large and real datasets, specifically GAP and CAP.

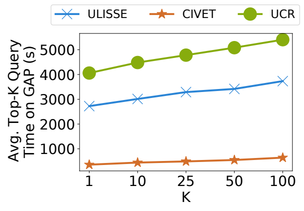
Performance of Top-K Matching. CIVET supports exact Top-K query as well. We also show the query time of the exact Top-K query varying K in Figure 9. Both ED and DTW distances are tested. The experimental results show that CIVET keeps a stable performance as the number of nearest neighbors increases.

Performance when Varying $l_{max} - l_{min}$. We explore the query time and the pruning power of different methods on GAP and CAP when varying the length range $l_{max} - l_{min}$ in Figure 10. The index pruning of CIVET (CIVET-ST1) outperforms that of ULISSE, and pruning during the scanning (CIVET-ST2) also exhibits superior capability compared to the pruning efficiency of UCR-US. For CIVET, due to the effective grouping strategies of subsequences, subsequences in the same blocks and envelopes have more similar features. Therefore, the pruning power drops with a slower trend, and the query time grows more slowly.

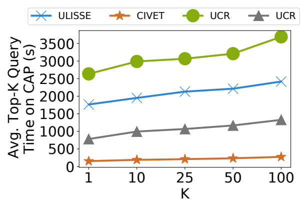
Accuracy of Approximate Matching In this part, we show the accuracy of the approximate matching compared with ULISSE. For ULISSE, we get the results from scaled queries of all possible lengths. We record how many approximate results belong to the



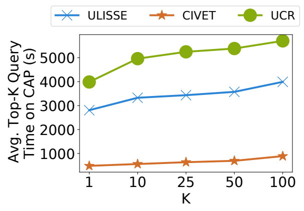
(a) Top-K Matching on GAP with ED



(b) Top-K Matching on GAP with DTW

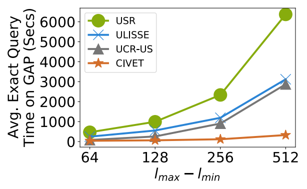


(c) Top-K Matching on CAP with ED

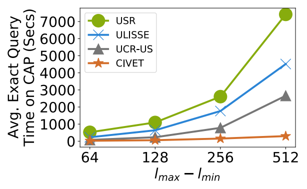


(d) Top-K Matching on CAP with DTW

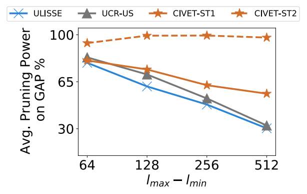
Figure 9: Exact Top-K Matching



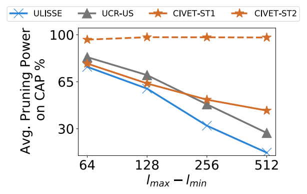
(a) Efficiency on GAP (ED)



(b) Efficiency on CAP (ED)

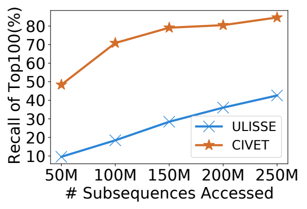


(c) Pruning Power on GAP (ED)

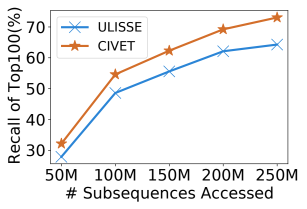


(d) Pruning Power on CAP (ED)

Figure 10: Exact Top-1 Matching Varying $l_{max} - l_{min}$



(a) Accuracy on GAP



(b) Accuracy on CAP

Figure 11: Approximate Matching

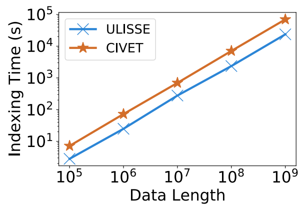
top-100 exact results (Recall of Top-100). Figure 11 demonstrates how the accuracy of approximate matching changes as the number of visited subsequences increases. CIVET achieves a higher recall on both datasets than ULISSE.

6.4 Scalability

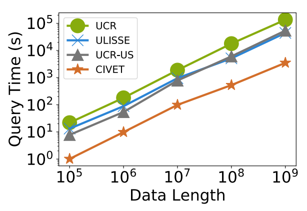
To test the scalability of CIVET, we conduct experiments on SYN with different sizes from 10^5 to 10^9. We record the index building time and the query time of CIVET in Figure 12. The time of CIVET building increases linearly with the amount of data. The efficient indexing algorithm proposed by ULISSE is also suitable for our index. So we can construct CIVET with the same time complexity (with a few more constants). In addition, we also depict the exact matching time of different methods. As the size of the dataset grows, the query time of our method grows linearly. Again, CIVET is faster than baselines by about one order of magnitude. Therefore, CIVET has the ability to index and query efficiently with great scalability.

7 RELATED WORK

Fixed-length Subsequence Matching. UCR SUITE [30] devise several techniques for computing lower bounds for the efficient similarity query. Many works have improved time series indexing through various summarization techniques [3, 11, 34, 37]. However, all of these indexes require queries with a preset and fixed length.



(a) Efficiency of Index Building



(b) Exact Top-1 Matching with ED

Figure 12: Scalability

Recent research has tried to tackle this limitation. Some studies propose efficient approaches to support constrained normalized subsequence matching, which is different from our problem. [4, 39]. ULISSE extends iSAX with envelope summarization so that it can index subsequences of variable length [25]. However, the methods above only support equal-length queries, meaning that the lengths of matched subsequences and queries are the same. As a result, they are not good at handling global scaling in the time dimension.

Variable-Length Subsequence Matching. The SpADe distance function is an elastic distance function that can handle time drift, amplitude drift, and shape scaling [6]. Lian et al. create the multi-scaled segment mean (MSM), which can be computed gradually and is suited to the stream features [23]. Kotsifakos et al. modify the edit distance for query by humming problem [21, 22]. These elastic measures do allow for variation in length. However, they only provide tolerance to local shifts in two sequences without the ability to deal with large global scaling. Recently, the sketch-based method has attracted the attention of many researchers [27, 28, 35]. While these methods are capable of matching variable-length data, their main focus is on the whole matching.

Uniform Scaling. Keogh et al. [18] utilize R-tree to index and accelerate uniform scaling distance calculation for whole matching. Ada et al. motivate the accommodation of US and DTW [12]. The extended work on UCR Suite proposes a lower bound distance to accelerate sequential match under uniform scaling [31]. A recent work refines the distance with a tighter bound [33]. However, these methods have not thoroughly investigated the properties of Uniform Scaling, and they have not been meticulously designed for subsequence queries under normalized conditions. There is still significant room for improvement in terms of efficiency.

8 CONCLUSION

In this paper, we extend the PAA as UPAA with uniform segmentation, which possesses the ability to tolerate global scaling. Besides combining two grouping strategies, our approach constructs a compact and effective index structure that supports efficient subsequence matching facilitated by index pruning and data filtering. Experimental results on different synthetic and real datasets demonstrate the efficiency, scalability, and effectiveness of our approach.

ACKNOWLEDGMENTS

The authors thank the anonymous reviewers for their insightful comments and suggestions. This work was supported by the National Key R&D Program of China (No. 2021YFB3300502).

REFERENCES

[1] Steven B. Achelis. 2001. *Technical Analysis from A to Z* (2nd ed.). McGraw Hill Professional.

[2] Alessandro Camerra, Themis Palpanas, Jin Shieh, and Eamonn Keogh. 2010. iSAX 2.0: Indexing and Mining One Billion Time Series. In *Proceedings of the 2010 IEEE International Conference on Data Mining (ICDM)*. IEEE Computer Society, 58–67.

[3] Alessandro Camerra, Jin Shieh, Themis Palpanas, Thanawin Rakthanmanon, and Eamonn Keogh. 2014. Beyond one billion time series: indexing and mining very large time series collections with *i*sax2+. *Knowledge and information systems* 39, 1 (2014), 123–151.

[4] 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.

[5] Georgios Chatzigeorgakidis, Dimitrios Skoutas, Kostas Patroumpas, Themis Palpanas, Spiros Athanasiou, and Spiros Skiadopoulos. 2023. Efficient Range and kNN Twin Subsequence Search in Time Series. *IEEE Transactions on Knowledge and Data Engineering* 35, 6 (2023), 5794–5807.

[6] Yueguo Chen, Mario A Nascimento, Beng Chin Ooi, and Anthony KH Tung. 2007. Spade: On shape-based pattern detection in streaming time series. In *2007 IEEE 23rd International conference on data engineering (ICDE)*. IEEE, 786–795.

[7] Hoang Anh Dau, Anthony Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, and Eamonn Keogh. 2019. The UCR time series archive. *IEEE/CAA Journal of Automatica Sinica* 6, 6 (2019), 1293–1305.

[8] Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn Keogh. 2008. Querying and mining of time series data: experimental comparison of representations and distance measures. *Proceedings of the VLDB Endowment* 1, 2 (2008), 1542–1552.

[9] Robert D Edwards, John Magee, and WH Charles Bassetti. 2018. *Technical analysis of stock trends*. CRC press.

[10] Waiyawuth Euachongprasit and Chotirat Ann Ratanamahatana. 2008. Efficient multimedia time series data retrieval under uniform scaling and normalisation. In *Advances in Information Retrieval: 30th European Conference on IR Research (ECIR)*. Springer, 506–513.

[11] Christos Faloutsos, Mudumbai Ranganathan, and Yannis Manolopoulos. 1994. Fast subsequence matching in time-series databases. *Acm Sigmod Record* 23, 2 (1994), 419–429.

[12] Ada Wai-Chee Fu, Eamonn Keogh, Leo Yung Hang Lau, Chotirat Ann Ratanamahatana, and Raymond Chi-Wing Wong. 2008. Scaling and time warping in time series querying. *The VLDB Journal* 17, 4 (2008), 899–921.

[13] Søren Kejser Jensen, Torben Bach Pedersen, and Christian Thomsen. 2017. Time series management systems: A survey. *IEEE Transactions on Knowledge and Data Engineering* 29, 11 (2017), 2581–2600.

[14] Markelle Kelly, Rachel Longjohn, and Kolby Nottingham. 2021. The UCI Machine Learning Repository. https://archive.ics.uci.edu

[15] Eamonn Keogh. 2006. A decade of progress in indexing and mining large time series databases. In *Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB)*. VLDB Endowment, 1268–1268.

[16] 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, 3 (2001), 263–286.

[17] Eamonn Keogh and Shruti Kasetty. 2003. On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration. *Data Mining and Knowledge Discovery* 7 (2003), 349–371.

[18] Eamonn Keogh, Themistoklis Palpanas, Victor B Zordan, Dimitrios Gunopulos, and Marc Cardle. 2004. Indexing large human-motion databases. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*. 780–791.

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

[20] Haridimos Kondylakis, Niv Dayan, Kostas Zoumpatianos, and Themis Palpanas. 2018. Coconut: a scalable bottom-up approach for building data series indexes. *Proceedings of the VLDB Endowment* 11, 6 (2018), 677–690.

[21] Alexios Kotsifakos, Panagiotis Papapetrou, Jaakko Hollmén, and Dimitrios Gunopulos. 2011. A subsequence matching with gaps-range-tolerances framework: a query-by-humming application. *Proceedings of the VLDB Endowment* 4, 11 (2011), 761–771.

[22] Alexios Kotsifakos, Panagiotis Papapetrou, Jaakko Hollmén, Dimitrios Gunopulos, Vassilis Athitsos, and George Kollios. 2012. Hum-a-song: a subsequence matching with gaps-range-tolerances query-by-humming system. *Proceedings of the VLDB Endowment* 5, 12 (2012), 1930–1933.

[23] Xiang Lian, Lei Chen, Jeffrey Xu Yu, Guoren Wang, and Ge Yu. 2007. Similarity match over high speed time-series streams. In *2007 IEEE 23rd International Conference on Data Engineering (ICDE)*. IEEE, 1086–1095.

[24] Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Bill Chiu. 2003. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD)*. ACM, 2–11.

[25] 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.

[26] Michele Linardi, Yan Zhu, Themis Palpanas, and Eamonn Keogh. 2018. Matrix Profile X: VALMOD - Scalable Discovery of Variable-Length Motifs in Data Series. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD)*. ACM, 1053–1066.

[27] Miro Mannino and Azza Abouzied. 2018. Expressive Time Series Querying with Hand-Drawn Scale-Free Sketches. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI)*. ACM, 1–13.

[28] Prithiviraj K Muthumanickam, Katerina Vrotsou, Matthew Cooper, and Jimmy Johansson. 2016. Shape grammar extraction for efficient query-by-sketch pattern matching in long time series. In *2016 IEEE Conference on Visual Analytics Science and Technology (VAST)*. IEEE, 121–130.

[29] John Paparrizos, Chunwei Liu, Aaron J. Elmore, and Michael J. Franklin. 2020. Debunking Four Long-Standing Misconceptions of Time-Series Distance Measures. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD)*. ACM, 1887–1905.

[30] 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 (KDD)*. ACM, 262–270.

[31] 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* 7, 3 (2013), 1–31.

[32] 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.

[33] Yilin Shen, Yanping Chen, Eamonn Keogh, and Hongxia Jin. 2018. Accelerating time series searching with large uniform scaling. In *Proceedings of the 2018 SIAM International Conference on Data Mining (ICDM)*. SIAM, 234–242.

[34] Jin Shieh and Eamonn Keogh. 2008. iSAX: indexing and mining terabyte sized time series. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 623–631.

[35] Tarique Siddiqui, Paul Luh, Zesheng Wang, Karrie Karahalios, and Aditya Parameswaran. 2020. Shapesearch: A flexible and efficient system for shape-based exploration of trendlines. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD)*. ACM, 51–65.

[36] Mario Giovanni Terzano, Liborio Parrino, Adriano Sherieri, Ronald Chervin, Sudhansu Chokroverty, Christian Guilleminault, Max Hirshkowitz, Mark Mahowald, Harvey Moldofsky, Agostino Rosa, et al. 2001. Atlas, rules, and recording techniques for the scoring of cyclic alternating pattern (CAP) in human sleep. *Sleep medicine* 2, 6 (2001), 537–554.

[37] Yang Wang, Peng Wang, Jian Pei, Wei Wang, and Sheng Huang. 2013. A data-adaptive and dynamic segmentation index for whole matching on time series. *Proceedings of the VLDB Endowment* 6, 10 (2013), 793–804.

[38] Zeyu Wang, Qitong Wang, Peng Wang, Themis Palpanas, and Wei Wang. 2023. Dumpy: A compact and adaptive index for large data series collections. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–27.

[39] 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 *IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 866–877.

[40] Byoung-Kee Yi and Christos Faloutsos. 2000. Fast Time Sequence Indexing for Arbitrary Lp Norms. In *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB)*. Morgan Kaufmann Publishers Inc., 385–394.

[41] 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 (SIGMOD)*. ACM, 181–192.

[42] Kostas Zoumpatianos and Themis Palpanas. 2018. Data series management: Fulfilling the need for big sequence analytics. In *IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 1677–1678.