Noname manuscript No.
(will be inserted by the editor)

Scalable Data Series Subsequence Matching with ULISSE

Michele Linardi · Themis Palpanas

Received: date / Accepted: date

Abstract Data series similarity search is an important operation and at the core of several analysis tasks and applications related to data series collections. Despite the fact that data series indexes enable fast similarity search, all existing indexes can only answer queries of a single length (fixed at index construction time), which is a severe limitation. In this work, we propose *ULISSE*, the first data series index structure designed for answering similarity search queries of *variable length* (within some range). Our contribution is two-fold. First, we introduce a novel representation technique, which effectively and succinctly summarizes multiple sequences of different length. Based on the proposed index, we describe efficient algorithms for approximate and exact similarity search, combining disk based index visits and in-memory sequential scans. Our approach supports non Z-normalized and Z-normalized sequences, and can be used with no changes with both Euclidean Distance and Dynamic Time Warping, for answering both *k-NN* and ϵ-range queries. We experimentally evaluate our approach using several synthetic and real datasets. The results show that *ULISSE* is several times, and up to orders of magnitude more efficient in terms of both space and time cost, when compared to competing approaches.

1 Introduction

Motivation. Data sequences are one of the most common data types, and they are present in almost every scientific and social domain (example application domains include meteorology, astronomy, chemistry, medicine, neuroscience, finance, agriculture, entomology, sociology, smart cities, marketing, operation health monitoring, human action recognition and others) [26,58,61,21,48]. This makes data series a data type of particular importance.

Informally, a data series (a.k.a data sequence, or time series) is defined as an ordered sequence of points, each one associated with a position and a corresponding value[1]. Recent advances in sensing, networking, data processing and storage technologies have significantly facilitated the processes of generating and collecting tremendous amounts of data sequences from a wide variety of domains at extremely high rates and volumes.

The *SENTINEL-2* mission [15] conducted by the European Space Agency (ESA) represents such an example of massive data series collection. The two satellites of this mission continuously capture multi-spectral images, designed to give a full picture of earth's surface every five days at a resolution of *10m*, resulting in over five trillion different data series. Such recordings will help monitor at fine granularity the evolution of the properties of the surface of the earth, and benefit applications such as land management, agriculture and forestry, disaster control, humanitarian relief operations, risk mapping and security concerns.

Data series analytics. Once the data series have been collected, the domain experts face the arduous tasks of processing and analyzing them [75,52,6] in order to identify patterns, gain insights, detect abnormalities, and extract useful knowledge. Critical part of this process is the data series similarity search operation, which lies at the core of sev-

Michele Linardi
LIPADE, Université de Paris
E-mail: michele.linardi@parisdescartes.fr

Themis Palpanas
LIPADE, Université de Paris
E-mail: themis@mi.parisdescartes.fr

[1] If the dimension that imposes the ordering of the sequence is time then we talk about time series. Though, a series can also be defined over other measures (e.g., angle in radial profiles in astronomy, mass in mass spectroscopy in physics, etc.). We use the terms *data series*, *time series*, and *sequence* interchangeably.

eral analysis and machine learning algorithms (e.g., clustering [46], classification [42], outliers [60,7,8], and others).

However, similarity search in very large data series collections is notoriously challenging [70,49,50,50,18,17,13,14,2], due to the high dimensionality (length) of the data series. In order to address this problem, a significant amount of effort has been dedicated by the data management research community to data series indexing techniques [51,13,14], which lead to fast and scalable similarity search [16,56,29,4,62,24,66,11,12,71,72,68,69,53,55,54,9,31,32,33].

Predefined constraints. Despite the effectiveness and benefits of the proposed indexing techniques, which have enabled and powered many applications over the years, they are restricted in different ways: either they only support similarity search with queries of a fixed size, or they do not offer a scalable solution. The solutions working for a fixed length, require that this length is chosen at index construction time (it should be the same as the length of the series in the index).

Evidently, this is a constraint that penalizes the flexibility needed by analysts, who often times need to analyze patterns of slightly different lengths (within a given data series collection) [24,25,57,39,40,41,44,52,6]. This is true for several applications. For example, in the *SENTINEL-2* mission data, oceanographers are interested in searching for similar coral bleaching patterns[2] of different lengths; at Airbus[3] engineers need to perform similarity search queries for patterns of variable length when studying aircraft take-offs and landings [47]; and in neuroscience, analysts need to search in Electroencephalogram (EEG) recordings for Cyclic Alternating Patterns (CAP) of different lengths (duration), in order to get insights about brain activity during sleep [3]. In these applications, we have datasets with a very large number of fixed length data series, on which analysts need to perform a large number of ad hoc similarity queries of (slightly) different lengths (as shown in Figure 1).

A straightforward solution for answering such queries would be to use one of the available indexing techniques. However, in order to support (exact) results for variable-length similarity search, we would need to (i) create several distinct indexes, one for each possible query length; and (ii) for each one of these indexes, index all overlapping subsequences (using a sliding window). We illustrate this in Figure 1, where we depict two similarity search queries of different lengths (ℓ and ℓ'). Given a data series from the collection, D_i (shown in black), we draw in red the subsequences that we need to compare to each query in order to compute the exact answer. Using an indexing technique implies inserting all the subsequences in the index: since we want to

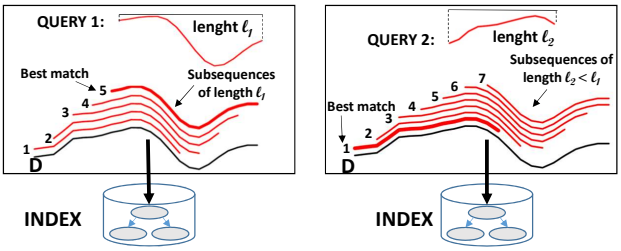


Fig. 1 Indexing for supporting queries of 2 different lengths.

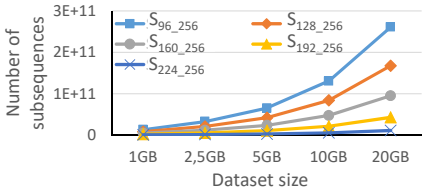


Fig. 2 Search space evolution of variable length similarity search. Each dataset contains series of length 256

answer queries of two different lengths, we are obliged to use two distinct indexes.

Nevertheless, this solution is prohibitively expensive, in both space and time. Space complexity is increased, since we need to index a large number of subsequences for each one of the supported query lengths: given a data series collection $C = D^1, ..., D^{|C|}$ and a query length range $[\ell_{min}, \ell_{max}]$, the number of subsequences we would normally have to examine (and index) is:

$S_{\ell_{min},\ell_{max}} = \sum_{\ell=1}^{(\ell_{max}-\ell_{min})+1} \sum_{i=1}^{|C|} (|D^i| - (\ell - 1))$.

Figure 2 shows how quickly this number explodes as the dataset size and the query length range increase: considering the largest query length range (S_{96-256}) in the 20GB dataset, we end up with a collection of subsequences (that need to be indexed) 5 orders of magnitude larger than the original dataset! Computational time is significantly increased as well, since we have to construct different indexes for each query length we wish to support.

In the current literature, a technique based on multi-resolution indexes [25,24] has been proposed in order to mitigate this explosion in size, by creating a smaller number of distinct indexes and performing more post-processing. Nonetheless, this solution works exclusively for *non* Z-normalized series[4] (which means that it cannot return results with similar trends, but different absolute values), and thus, renders the solution useless for a wide spectrum of applications. Besides, it only mitigates the problem, since it still leads to a space explosion (albeit, at a lower rate), and therefore, it is not scalable, either.

We note that the technique discussed above (despite its limitations) is indeed the current state of the art, and no

[2] http://www.esa.int/Our_Activities/Observing_the_Earth/

[3] http://www.airbus.com/

[4] Z-normalization transforms a series so that it has a mean value of zero, and a standard deviation of one. This allows similarity search to be effective, irrespective of shifting (i.e., offset translation) and scaling[28].

other technique has been proposed since, even though during the same period of time we have witnessed lots of activity and a steady stream of papers on the *single-length* similarity search problem (e.g., [29,4,62,10,66,11,71,72,68,69, 53,55,54,31,32,33]). This attests to the challenging nature of the problem we are tackling in this paper.

Contributions. In this work, we propose *ULISSE* (ULtra compact Index for variable-length Similarity SEarch in data series), which is the first single-index solution that supports fast approximate and exact answering of variable-length (within a given range) similarity search queries for both non Z-normalized and Z-normalized data series collections. Our approach can be used with no changes with both Euclidean Distance and Dynamic Time Warping, for answering both *k-NN* and ϵ-range queries.

ULISSE produces exact (i.e., correct) results, and is based on the following key idea: a data structure that indexes data series of length ℓ, already contains all the information necessary for reasoning about any subsequence of length $\ell' < \ell$ of these series. Therefore, the problem of enabling a data series index to answer queries of variable-length, becomes a problem of how to reorganize this information that already exists in the index. To this effect, *ULISSE* proposes a new summarization technique that is able to represent contiguous and overlapping subsequences, leading to succinct, yet powerful summaries: it combines the representation of several subsequences within a single summary, and enables fast (approximate and exact) similarity search for variable-length queries.

Our contributions can be summarized as follows[5]:

- We introduce the problem of Variable-Length Subsequences Indexing, which calls for a single index that can inherently answer queries of different lengths.
- We provide a new data series summarization technique, able to represent several contiguous series of different lengths. This technique produces succinct, discretized envelopes for the summarized series, and can be applied to both non Z-normalized and Z-normalized data series.
- Based on this summarization technique, we develop an indexing algorithm, which organizes the series and their discretized summaries in a hierarchical tree structure, namely, the *ULISSE* index.
- We propose efficient exact and approximate k-NN algorithms, suitable for the *ULISSE* index, which can compute similarity using either Euclidean Distance or Dynamic Time Warping measure.
- Finally, we perform an experimental evaluation with several synthetic and real datasets. The results demonstrate the effectiveness and scalability of *ULISSE* to dataset sizes that competing approaches cannot handle.

[5] A preliminary version of this work has appeared elsewhere [38, 37].

Paper Organization. The rest of this paper is organized as follows. Section 2 discusses related work, and Section 3 formulates the problem. In Section 4, we describe the *ULISSE* summarization techniques, and in Sections 5 and 6 we explain our indexing and query answering algorithms. Section 7 describes the experimental evaluation, and we conclude in Section 8.

2 Related Work

Data series indexes. The literature includes several techniques for data series indexing [13], which are all based on the same principle: they first reduce the dimensionality of the data series by applying some summarization technique (e.g., Piecewise Aggregate Approximation (PAA) [27], or Symbolic Aggregate approXimation (SAX) [62]. However, all the approaches mentioned above share a common limitation: they can only answer queries of a fixed, predetermined length, which has to be decided before the index creation.

Faloutsos et al. [16] proposed the first indexing technique suitable for variable length similarity search query. This technique extracts subsequences that are grouped in MBRs (Minimum Bounding Rectangles) and indexed using an R-tree. We note that this approach works only for non Z-normalized sequences. An improvement of this approach was proposed by Kahveci and Singh [25]. They described MRI (Multi Resolution Index), which is a technique based on the construction of multiple indexes for variable length similarity search query.

Storing subsequences at different resolutions (building indexes for different series lengths) provided a significant improvement over the earlier approach, since a greater part of a single query is considered during the search. Subsequently, Kadiyala and Shiri [24] redesigned the MRI construction, in order to decrease the indexing size and construction time. This new indexing technique, called Compact Multi Resolution Index (CMRI), has a space requirement, which is 99% smaller than the one of MRI. The authors also redefined the search algorithm, guaranteeing an improvement of the range search proposed upon the MRI index.

Loh et al. [43] proposed Index Interpolation for variable length subsequence similarity search for ϵ-range queries. This approach uses a single index that supports ϵ-range search for subsequences of a fixed length that is smaller than the query length. The search starts by considering a query prefix subsequence of the same length as the one supported by the index. During this process, the algorithm computes the distances between the original query and candidates of the same length, if the prefixes of these candidates have a distance to the query prefix smaller than the proposed bound. The authors proved the correctness of their solution, showing that their bounding strategy provides correct results

for both non Z-normalized and Z-normalized subsequences. We note that, based on the ϵ-range search, this approach can also answer k-NN queries, thanks to the framework proposed by Han et al. [19].

More recently, Wu et al. [67] have proposed the KV-Match index, which supports ϵ-range similarity search queries of variable length, using both Z-normalized Euclidean and DTW distances. The idea of this technique is similar to the CMRI one, since many indexes are built for different subsequence window lengths, which are considered at query time using multiple query segments. We note that for Z-normalized sequences, this method provides exact answers only for *constrained* ϵ-range search. To this effect, two new parameters that constrain the mean and the standard deviation of a valid result are considered at query answering time.

In contrast to CMRI and KV-Match, our approach uses a single index that is able to answer similarity search queries of variable length over larger datasets, and works for both non Z-normalized and Z-normalized series (a feature that is not supported by any of the previously introduced indexing techniques).

Sequential scan techniques. Even though recent works have shown that sequential scans can be performed efficiently [57,45], such techniques are mostly applicable when the dataset consists of a single, very long data series, and queries are looking for potential matches in small subsequences of this long data series. Such approaches, in general, do not provide any benefit when the dataset is composed of a large number of small data series, like in our case. Therefore, indexing is required in order to efficiently support data exploration tasks, which involve ad-hoc queries, i.e., the query workload is not known in advance.

3 Preliminaries and Problem Formulation

Let a data series $D = d_1,...,d_{|D|}$ be a sequence of numbers $d_i \in \mathbb{R}$, where $i \in \mathbb{N}$ represents the position in D. We denote the length, or size of the data series D with $|D|$. The subsequence $D_{o,\ell}$=$d_o,...,d_{o+\ell-1}$ of length ℓ, is a contiguous subset of ℓ points of D starting at offset o, where $1 \leq o \leq |D|$ and $1 \leq \ell \leq |D| - o + 1$. A subsequence is itself a data series. A data series collection, C, is a set of data series.

We say that a data series D is Z-normalized, denoted D^n, when its mean μ is 0 and its standard deviation σ is 1. The normalized version of $D = d_1, ..., d_{|D|}$ is computed as follows: $D^n = \{\frac{d_1-\mu}{\sigma}, ..., \frac{d_{|D|}-\mu}{\sigma}\}$. Z-normalization is an essential operation in several applications, because it allows similarity search irrespective of shifting and scaling [28,57].

Euclidean Distance. Given two data series $D = d_1, ..., d_{|D|}$ and $D' = d'_1, ..., d'_{|D'|}$ of the same length (i.e., $|D| = |D'|$), we can calculate their Euclidean Distance as follows:

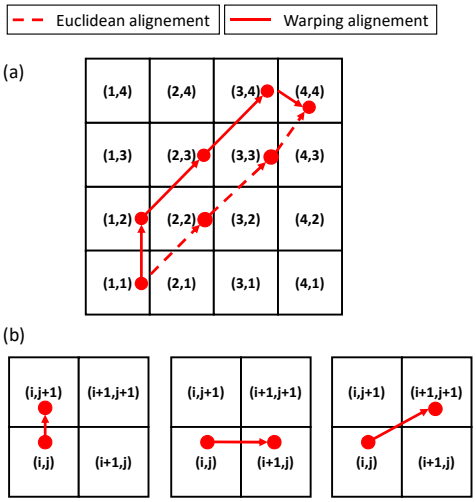


Fig. 3 *(a)* Euclidean and Warping alignment in a squared matrix. *(b)* Valid index steps in a warping alignment.

$ED(D, D') = \sqrt{\sum_i^{|D|} d(d_i, d'_i)}$, where the distance function d is applied to two real values, namely A and B, as follows: $d(A, B) = (A - B)^2$.

Dynamic Time Warping. The Euclidean distance is a lock-step measure, which is computed by summing up the distances between pairs of points that have the same positions in their respective series. Dynamic Time Warping (DTW) [34] represents a more elastic measure, allowing for small mis-alignments of the matched points on the x-axis.

Given two data series D and D', the DTW distance is computed by considering the differences between pairs of points $(d(d_i, d'_j))$, where the indexes i, j might be different. In this manner, a particular alignment of D and D' is performed before to compute the distance. We define a sequence alignment as a vector of index pairs $A \in \mathbb{R}^{\ell \times 2}$, where $(i, j) \in A \iff 1 \leq i, j \leq \ell$, and ℓ is the length of the two series. The alignment of the Euclidean Distance is a special case, where the indexes are equal to their position in A. In the case of two series of length ℓ, the space of the possible alignments spans the paths that join two cells in a squared matrix composed by ℓ^2 cells. In Figure 3(a), we depict a Euclidean distance alignment of two series of length *4*, which exactly crosses the diagonal of the matrix, joining the cells *(1,1)* and *(4,4)*. On the other hand, in the same figure we report another possible alignment that we call warping alignment, which deviates from the diagonal. We use the terms *warping path* and *warping alignment* interchangeably.

In order to restrict the allowed paths, we can apply the following local constraints on the index pairs:

– We require that the first and last pairs of A correspond to the first and last pairs of points in D and D', respectively. If $|D| = |D'| = \ell$, we have $A[1] = (1, 1)$

and $A[\ell] = (\ell, \ell)$. Furthermore, for any $(a, b), (c, d) \in A \iff (a \neq c) \vee (b \neq d)$. This latter, avoids to consider the same index pair twice in a single path.

- Given $k \in \mathbb{N}$ $(1 < k \leq \ell)$, we require that $A[k][0] - A[k-1][0] \leq 1$ and $A[k][1] - A[k-1][1] \leq 1$ always holds. This restricts each index to move by at most 1 unit to its next alignment position.
- Moreover, we always require that $A[k][0] - A[k-1][0] \geq 0$ and $A[k][1] - A[k-1][1] \geq 0$. This guarantees a monotonic movement of the path, towards the last index pair. In Figure 3(b), we depict the three possible steps that each index pair can perform in a valid alignment.

These constraints permit to bound the length of an alignment between two series of length ℓ, between ℓ and $2 \times \ell - 1$. Typically, warping paths are also subject to global constraints. We can thus set their maximum deviation from the matrix diagonal. In that regard, Sakoe and Chiba [59] and Itakura [23] proposed different warping path constraints, which restrict the matrix positions that a valid path can visit. The *Sakoe-Chiba band* [59] constraint allows each index of a warping path to be at most *r* points far from the diagonal (Euclidean Distance alignment). On the other hand, the *Itakura-parallelogram* [23] constraint allows to choose different r values depending on the index position i. In general, r is called the *warping window*.

Given a valid warping path, A^*, that satisfies the previously introduced constraints, we can formally define the DTW distance between two series D and D' of the same length ℓ, as:

$$DTW(D, D') = \underset{A^*}{\text{argmin}}(\sqrt{\sum_{i}^{|A^*|} d(d_{A^*[i][0]}, d'_{A^*[i][1]})}).$$

We note that computing the DTW distance corresponds to finding the valid alignment that minimizes the sum of the distances.

In Figure 4, we consider two series (D and D'), which are extracted from two offsets that are *5* points away, in the same long sequence. In this manner, the prefix of D is equal to the suffix of D', which starts at position *6*. In the plots, the values of D span the right vertical axis, whereas those of D' the left one. If we compute the Euclidean distance, as depicted in Figure 4(a), the fixed alignment of points does not capture the similarity of the two series. On the other hand, when computing the DTW distance, the warping path aligns the two similar parts, as reported in Figure 4(b). At the bottom of the figure, we also report the warping path, which is constrained by a *Sakoe-Chiba band*.

Problem Definition. The problem we wish to solve in this paper is the following:

Problem 1 (Variable-Length Subsequences Indexing) Given a data series collection C, and a series length range $[\ell_{min}, \ell_{max}]$, we want to build an index that supports exact similarity search, under the Euclidean and Dynamic Time Warping (DTW) measures, for queries of any length within the range $[\ell_{min}, \ell_{max}]$.

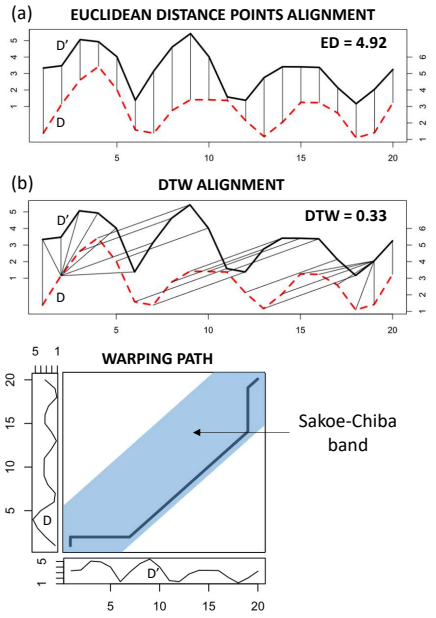


Fig. 4 *(a)* Euclidean distance alignment between the data series D and D'. *(b)* DTW Alignment between D and D'. In the bottom part, the path is depicted in the $|D| \times |D'|$ matrix, contoured by the Sakoe-Chiba band.

In our case similarity search is formally defined as follows:

Definition 1 (Similarity search) Given a data series collection $C = \{D^1, ..., D^C\}$, a series length range $[\ell_{min}, \ell_{max}]$, a query data series Q, where $\ell_{min} \leq |Q| \leq \ell_{max}$, and $k \in \mathbb{N}$, we want to find the set $R = \{D^i_{o,\ell} | D^i \in C \wedge \ell = |Q| \wedge (\ell + o - 1) \leq |D^i|\}$, where $|R| = k$. We require that $\forall D^i_{o,\ell} \in R \nexists D^{i'}_{o',\ell'}$ *s.t.* $dist(D^{i'}_{o',\ell'}, Q) < dist(D^i_{o,\ell}, Q)$, where $\ell' = |Q|$, $(\ell' + o' - 1) \leq |D^{i'}|$ and $D^{i'} \in C$. We informally call R, the k *nearest neighbors* set of Q. Given two generic series of the same length, namely D and D' the function $dist(d, d') = ED()$.

In this work, we perform Similarity Search using either Euclidean Distance (ED) or Dynamic Time Warping (DTW), as the $dist$ function.

3.1 The iSAX Index

The Piecewise Aggregate Approximation (PAA) of a data series D, $PAA(D) = \{p_1, ..., p_w\}$, represents D in a w-dimensional space by means of w real-valued segments of length s, where the value of each segment is the mean of the

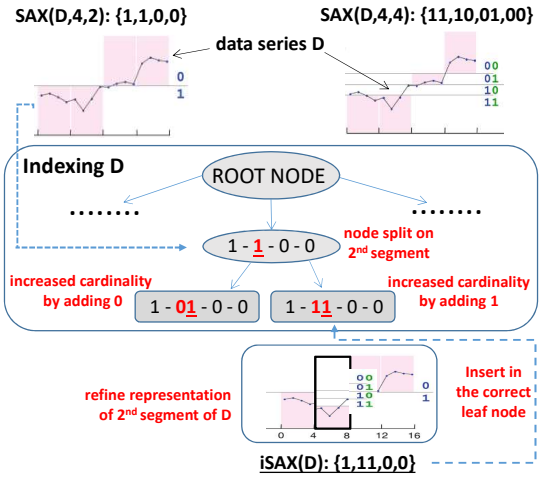


Fig. 5 Indexing of series D (and an inner node split).

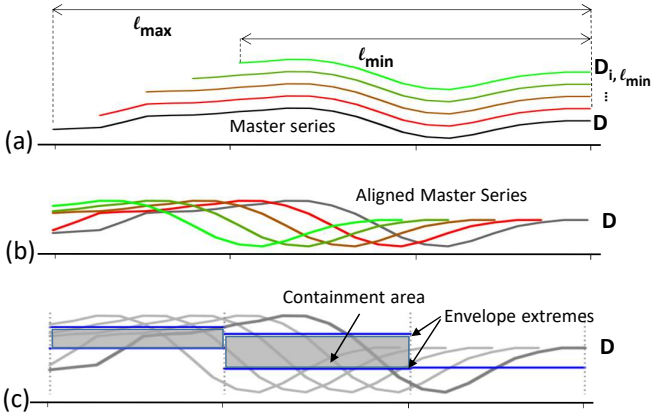


Fig. 6 *a)* master series of D in the length interval ℓ_{min}, ℓ_{max}. *b)* Zero-aligned master series. *c)* Envelope built over the master series.

corresponding values of D [27]. We denote the first k dimensions of $PAA(D)$, ($k \leq w$), as $PAA(D)_{1,..,k}$. Then, the $iSAX$ representation of a data series D, denoted by $SAX(D, w, |alphabet|)$, is the representation of $PAA(D)$ by w discrete coefficients, drawn from an alphabet of cardinality $|alphabet|$ [62].

The main idea of the $iSAX$ representation (see Figure 5, top), is that the real-values space may be segmented by $|alphabet| - 1$ breakpoints in $|alphabet|$ regions that are labeled by distinct symbols: binary values (e.g., with $|alphabet| = 4$ the available labels are $\{00, 01, 10, 11\}$). $iSAX$ assigns symbols to the PAA coefficients, depending in which region they are located.

The iSAX data series index is a tree data structure [62, 11], consisting of three types of nodes (refer to Figure 5). (i) The root node points to n children nodes (in the worst case $n = 2^w$, when the series in the collection cover all possible iSAX representations). (ii) Each inner node contains the iSAX representation of all the series below it. (iii) Each leaf node contains both the iSAX representation *and* the raw data of all the series inside it (in order to be able to prune false positives and produce exact, correct answers). When the number of series in a leaf node becomes greater than the maximum leaf capacity, the leaf splits: it becomes an inner node and creates two new leaves, by increasing the cardinality of one of the segments of its iSAX representation. The two refined iSAX representations (new bit set to *0* and *1*) are assigned to the two new leaves.

4 The ULISSE framework

The key idea of the *ULISSE* approach is the succinct summarization of *sets* of series, namely, overlapping subsequences. In this section, we present this summarization method.

4.1 Representing Multiple Subsequences

When we consider, contiguous and overlapping subsequences of different lengths within the range $[\ell_{min}, \ell_{max}]$ (Figure 6(a), we expect the outcome as a bunch of similar series, whose differences are affected by the misalignment and the different number of points. We conduct a simple experiment in Figure 6(b), where we zero-align all the series shown in Figure 6(a); we call those *master series*.

Definition 2 (Master Series) Given a data series D, and a subsequence length range $[\ell_{min}, \ell_{max}]$, the master series are subsequences of the form $D_{i,min(|D|-i+1,\ell_{max})}$, for each i such that $1 \leq i \leq |D| - (\ell_{min} - 1)$, where $1 \leq \ell_{min} \leq \ell_{max} \leq |D|$.

We observe that the following property holds for the master series.

Lemma 1 *For any master series of the form $D_{i,\ell'}$, we have that $PAA(D_{i,\ell'})_{1,..,k} = PAA(D_{i,\ell''})_{1,..,k}$ holds for each ℓ'' such that $\ell'' \geq \ell_{min}$, $\ell'' \leq \ell' \leq \ell_{max}$ and $\ell', \ell''\%k = 0$.*

Proof It trivially follows from the fact that, each non master series is always entirely overlapped by a master series. Since the subsequences are not subject to any scale normalization, their prefix coincides to the prefix of the equi-offset master series.

Intuitively, the above lemma says that by computing only the PAA of the master series in D, we are able to represent the PAA prefix of any subsequence of D.

When we zero-align the PAA summaries of the master series, we compute the minimum and maximum PAA values (over all the subsequences) for each segment: this forms what we call an *Envelope* (refer to Figure 6.c). (When the length of a master series is not a multiple of the PAA segment length, we compute the PAA coefficients of the longest prefix, which is multiple of a segment.) We call *containment area* the space in between the segments that define the Envelope.

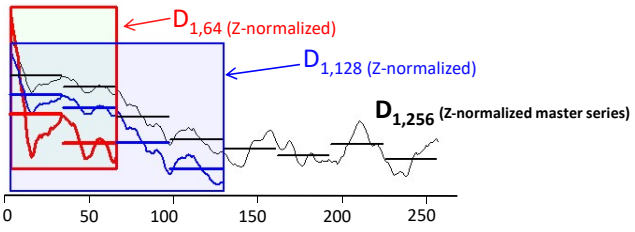


Fig. 7 Master series $D_{1,256}$ with marked PAA coefficients.

4.2 PAA Envelope for Non-Z-Normalized Subsequences

In this subsection, we formalize the concept of the *Envelope*, introducing a new series representation.

We denote by L and U the PAA coefficients, which delimit the lower and upper parts, respectively, of a containment area (see Figure 6.c). Furthermore, we introduce a parameter γ, which corresponds to the number of master series we represent by the Envelope. This allows to tune the number of subsequences of length in the range $[\ell_{min}, \ell_{max}]$, that a single Envelope represents, influencing both the tightness of a containment area and the size of the Index (number of computed Envelopes). We will show the effect of the relative tradeoff i.e., Tightness/Index size in the Experimental evaluation. Given a, the point from where we start to consider the subsequences in D, and s, the chosen length of the PAA segment, we refer to an Envelope using the following signature:

$$paaENV_{[D,\ell_{min},\ell_{max},a,\gamma,s]} = [L, U] \quad (1)$$

4.3 PAA Envelope for Z-Normalized Subsequences

So far we have considered that each subsequence in the input series D is not subject of any scale normalization, i.e., is not Z-normalized. We introduce here a negative result, concerning the *unsuitability* of a generic $paaENV_{[D,\ell_{min},\ell_{max},a,\gamma,s]}$ to describe subsequences that are Z-normalized.

Intuitively, we argue that the PAA coefficients of a single master series $D_{i,a}$, generate a containment area, which may not embed the coefficients of the Z-normalized subsequence in the form $D'_{i,a'}$, for $a' < a$. This happens, because Z-normalization causes the subsequences of different lengths to change their shape, and even shift on the y-axis. Figure 7 depicts such an example.

We can now formalize this negative result.

Lemma 2 *A $paaENV_{[D,\ell_{min},\ell_{max},a,\gamma,s]}$ is not guaranteed to contain all the PAA coefficients of the Z-normalized subsequences of lengths $[\ell_{min}, \ell_{max}]$, of D.*

Proof To prove the correctness of the lemma, it suffices to pick such a case where a subsequence of D, namely $D_{a,\ell'}$, with $\ell_{min} \leq \ell' \leq \ell_{max}$, is not encoded by $paaENV_{[D,\ell_{min},\ell_{max},a,\gamma,s]}$. Formally, we should consider the case where $\exists k$ such that $PAA(D_{i,\ell'})_k > U_k$ or $PAA(D_{i,\ell'})_k < L_k$. We may pick a Z-normalized series D choosing $\ell_{max} = |D| = \ell_{min} + 1$ and $\gamma = 0$. The resulting $paaENV_{[D,\ell_{min}=\ell_{max}-1,\ell_{max}=|D|,i=1,\gamma=0,s]}$ obtains equal bounds, namely $L = U$. Let consider the z-normalized subsequence $D_{1,\ell_{min}}$. Its PAA coefficients must be in the envelope. This implies that, $PAA(D_{1,\ell_{min}})_1 = L_1 = U_1$ must hold. If s is the PAA segment length, in the case of Z-normalization, $PAA(D_{1,\ell_{min}})_1 = (((\sum_{i=1}^{s} d_i) - (\mu_{D_{1,\ell_{min}}} \times s))/\sigma_{D_{1,\ell_{min}}})/s$ and $U_1 = (((\sum_{i=1}^{s} d_i) - (\mu_D \times s))/\sigma_D)/s$. Therefore, the following equation: $(\mu_{D_{1,\ell_{min}}} \times s)/\sigma_{D_{1,\ell_{min}}} = (\mu_D \times s)/\sigma_D$ holds, which is equivalent to $\mu_{D_{1,\ell_{min}}}/\sigma_{D_{1,\ell_{min}}} = \mu_D/\sigma_D$. At this point we may have that $\mu_D = \mu_{D_{1,\ell_{min}}}$, when $D_{\ell max,1} = \mu_{D_{1,\ell_{min}}}$. This clearly leads to have a smaller dispersion on D than $D_{1,\ell min}$ and thus $\sigma_D < \sigma_{D_{1,\ell_{min}}} \implies PAA(D_{1,\ell_{min}})_1 \neq L_1 \neq U_1$.

If we want to build an Envelope, containing all the Z-normalized sequences, we need to take into account the shifted coefficients of the Z-normalized subsequences, which are not master series. Hence, each PAA segment coefficient (in a master series) will be represented by the set of values resulting from the Z-normalizations of all the subsequences of length in $[\ell_{min}, \ell_{max}]$ that are not master series and contain that segment.

Given a generic master series $D_{i,\ell} = \{d_i, ..d_{i+\ell-1}\}$, and s the length of the segment, its k^{th} PAA coefficient set is computed by: $PAA^*(D_{i,\ell})_k = \{(\frac{(\sum_{p=s(k-1)+1}^{s(k-1)+s} d_p) - (\mu_{D_{i,\ell'}} \times s)}{\sigma_{D_{i,\ell'}}})/s | \ell_{min} \leq \ell' \leq \ell_{max}, \ell' \geq (s(k-1) + s - (i-1))\}$ (2).

In Figure 8, we depict an example of PAA^* computation for the first segment of the master series D.

We can then follow the same procedure as before (in the case of non Z-normalized sequences), computing the minimum and maximum PAA coefficients for each segment given by the above formula, in order to get the Envelope for the Z-normalized sequences (which we also denote with $paaENV$).

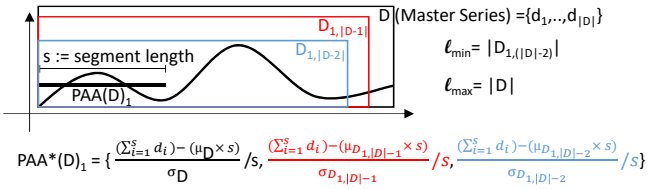


Fig. 8 $PAA^*(D)_1$ computation. Since the first PAA segment (of length s) of the master series D, is also the first one of the two non master series $D_{1,|D-1|}$, $D_{1,|D-2|}$, three PAA coefficients are computed with the different normalizations.

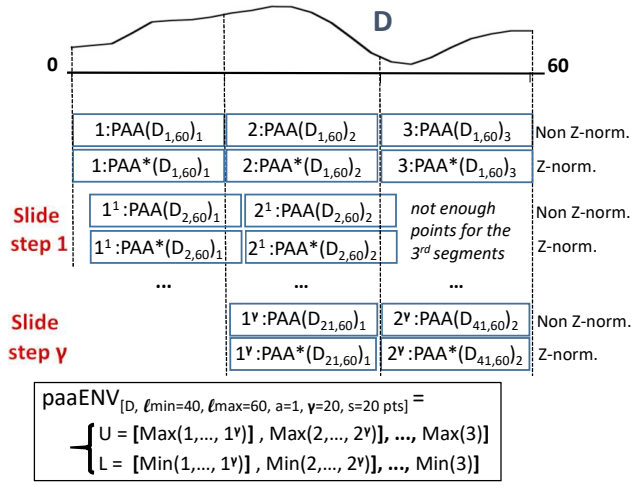


Fig. 9 $uENV$ building, with input: data series D of length 60, PAA segment size $= 20$, $\gamma = 20$, $\ell_{min} = 40$ and $\ell_{max} = 60$.

4.4 Indexing the Envelopes

Here, we define the procedure used to index the Envelopes. In that regard, we aim to adapt the $iSAX$ indexing mechanism (depicted in Figure 5).

Given a $paaENV$, we can translate its PAA extremes into the relative iSAX representation: $uENV_{paaENV_{[D,\ell_{min},\ell_{max},a,\gamma,s]}} = [iSAX(L), iSAX(U)]$, where $iSAX(L)$ ($iSAX(U)$) is the vector of the minimum (maximum) PAA coefficients of all the segments corresponding to the subsequences of D.

The *ULISSE* Envelope, $uENV$, represents the principal building block of the *ULISSE* index. Note that, we might remove for brevity the subscript containing the parameters from the $uENV$ notation, when they are explicit.

In Figure 9, we show a small example of envelope building, given an input series D. The picture shows the PAA coefficients computation of the master series. They are calculated by using a sliding window starting at point $a = 1$, which stops after γ steps. Note that the Envelope generates a containment area, which embeds all the subsequences of D of all lengths in the range $[\ell_{min}, \ell_{max}]$.

Algorithm 1: $uENV computation$

Input: **float**[] D, **int** s, **int** ℓ_{min}, **int** ℓ_{max}, **int** γ, **int** a
Output: $\mathbf{uENV[iSAX_{min}, iSAX_{max}]}$

```
1  int w ← ⌊ℓmax/s⌋ ;
2  int segUpdateList[S] ← {0,...,0};
3  float U[w] ← {−∞, ..., −∞}, L[w] ← {∞, ..., ∞};
4  if |D| − (i − 1) ≥ ℓmin then
5      float paaRSum ← 0;
       // iterate the master series.
6      for i ← a to min(|D|,a + ℓmax + γ) do
           // running sum of paa segment
7          paaRSum ← paaRSum + D[i];
8          if (j-a) > s then
9              paaRSum ← paaRSum - D[i-s];
10         for z ← 1 to min(⌊[i-(a-1)] / s⌋,w) do
11             if segUpdatedList[z] ≤ γ then
12                 segUpdateList[z] ++;
13                 float paa ← (paaRSum / s);
14                 L[z] ← min(paa, L[z]);
15                 U[z] ← max(paa, U[z]);
16     uENV ← [iSAX(L),iSAX(U)];
17 else
18     uENV ← ∅;
```

5 Indexing Algorithm

5.1 Indexing Non-Z-Normalized Subsequences

We are now ready to introduce the algorithms for building an $uENV$. Algorithm 1 describes the procedure for non-Z-normalized subsequences. As we noticed, maintaining the running sum of the last s points, i.e., the length of a PAA segment (refer to Line 7), allows us to compute all the PAA values of the expected envelope in $O(w(\ell_{max} + \gamma))$ time in the worst case, where $\ell_{max} + \gamma$ is the points window we need to take into account for processing each master series, and w is the number of PAA segments in the maximum subsequence length ℓ_{max}. Since w, is usually a very small number (ranging between 8-16), it essentially plays the role of a constant factor. In order to consider not more than γ steps for each segment position, we store how many times we use it, to update the final envelope in the vector, in Line 2.

5.2 Indexing Z-Normalized Subsequences

In Algorithm 2, we show the procedure that computes an indexable Envelope for Z-normalized sequences, which we denote as $uENV_{norm}$. This routine iterates over the points of the overlapping subsequences of variable length (*First loop* in Line 7), and performs the computation in two parts. The first operation consists of computing the sum of each PAA segment we keep in the vector $PAAs$ defined in Line 2. When we encounter a new point, we update the sum of all the segments that contain that point (Lines 8-11). The second part, starting in Line 16 (*Second loop*), performs the segment normalizations, which depend on the statistics (mean and std.deviation) of all the subsequences of different length (master and non-master series), in which they appear. During this step, we keep the sum and the squared sum of

Algorithm 2: $uENV_{norm}$ computation

```
   Input: float[] D, int s, int ℓ_min, int ℓ_max, int γ, int a
   Output: uENV_norm[iSAX_min, iSAX_max]
1  int w ← ⌊ℓ_max/s⌋ ;
   // sum of PAA segments values
2  float PAAs [ℓ_max + γ − (s − 1)] ← {0,...,0};
3  float U[w] ← {−∞, ..., −∞}, L[w] ← {∞, ..., ∞};
4  if |D| − (a − 1) ≥ ℓ_min then
5      int nSeg← 1;
6      float accSum, accSqSum ← 0;
       // First loop: Iterate the points.
7      for i ← a to min(|D|,(a+ℓ_max+γ)) do
           // update sum of PAA segments values
8          if i − a > s then
9              nSeg++;
10             PAAs[nSeg] ← PAAs[nSeg-1] - D[i-s];
11         PAAs[nSeg] += D[i];
           // keep sum and squared sum.
12         accSum += D[i], accSqSum += (D[i])^2;
           // the window contains enough points.
13         if i-(a-1) ≥ ℓ_min then
14             acSAc ← accSum, acSqSAc ← accSqSum;
15             int nMse ← min(γ+1,(i-(a-1)-ℓ_min) + 1);
               // Normalizations of PAA coefficients.
16             for j ← 1 to nMse do
17                 int wSubSeq ← i-(a-1)-(j-1) ;
18                 if wSubSeq ≤ ℓ_max then
19                     float μ ←acSAc/wSubSeq;
20                     float σ ← sqrt((acSqSAc/wSubSeq − μ^2));
21                     int nSeg ← ⌊wSubSeq÷s⌋;
22                     for z ← 1 to nSeg do
23                         float a ← PAAs[j+[(z-1)×s]];
24                         float b ← s×μ;
25                         float paaNorm ← ((a−b)/σ)/s;
26                         L[z] ← min(paaNorm, L[z]);
27                         U[z] ← max(paaNorm, U[z]);
28                 acSAc -= D[j], acSqSAc -= (D[j])^2;
29     uENV_norm ← [iSAX(L),iSAX(U)];
30 else
31     uENV_norm ← ∅;
```

the window, which permits us to compute the mean and the standard deviation in constant time (Lines 19,20). We then compute the Z-normalizations of all the PAA coefficients in Line 25, by using Equation 2.

In Figure 10, we show an example that illustrates the operation of the algorithm. In *1*, the *First loop* has iterated over *8* points (marked with the dashed square). Since they form a subsequence of length ℓ_{min}, the *Second loop* starts to compute the Z-normalized PAA coefficients of the two segments, computing the mean and the standard deviation using the sum ($acSAc$) and squared sum ($acSqAc$) of the points considered by the *First loop* (gray circles). The second step takes place after that the *First loop* has considered the 9^{th} point (black circle) of the series. Here, the *Second loop* updates the sum and the squared sum, with the new point, calculating then the corresponding new Z-normalized PAA coefficients. At step *3*, the algorithm considers the second subsequence of length ℓ_{min}, which is contained in the nine points window. The *Second loop* considers in order all the overlapping subsequences, with different prefixes and length. This permits to update the statistics (and all possible normalizations) in constant time. The algorithm terminates, when all the points are considered by the *First loop*, and the

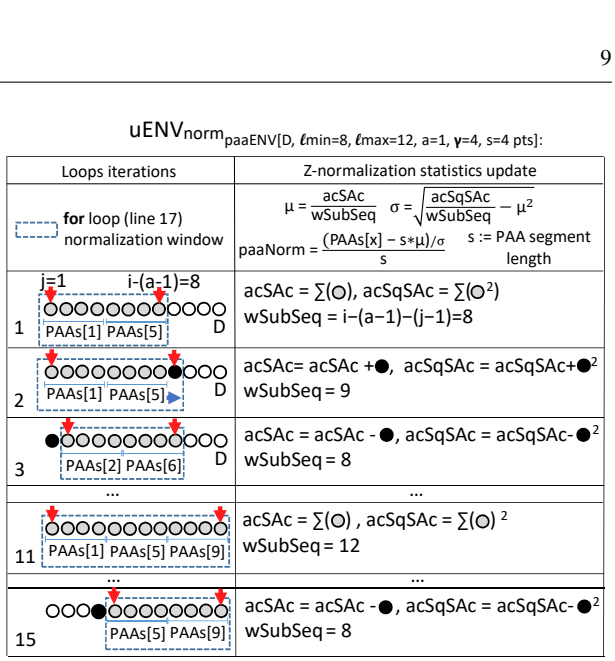


Fig. 10 Running example of Algorithm 2. *Left column*) Points iteration, the dashed squared contours the subsequence used to normalize the PAA coefficients in the Second loop. *Right column*) Statistics update at each step, which serve the computation of μ and σ of each possible coefficients normalization.

Algorithm 3: *ULISSE* index computation

```
   Input: Collection C, int s, int ℓ_min, int ℓ_max, int γ, bool bNorm
   Output: ULISSE index I
1  foreach D in C do
2      int a' ← ∅;
3      uENV E ← ∅;
4      while true do
5          if bNorm then
6              E ← uENV_norm(D, s, ℓ_min, ℓ_max, γ, a');
7          else
8              E ← uENV(D, s, ℓ_min, ℓ_max, γ, a');
9          a' ← a' + γ + 1 ;
10         if E == ∅ then
11             break;
12         bulkLoadingIndexing(I, E);
13         I.inMemoryList.add(maxCardinality(E));
```

Second loop either encounters a subsequence of length ℓmin (as depicted in the step *15*), or performs at most γ iterations, since all the subsequences starting at position $a + \gamma + 1$ or later (if any) will be represented by other Envelopes.

5.2.1 Complexity Analysis

Given w, the number of PAA segments in the window of length ℓ_{max}, and $M = \ell_{max} - \ell_{min} + \gamma$, the number of master series we need to consider, building a normalized Envelope, $uENV_{norm}$, takes $O(M\gamma w)$ time.

5.3 Building the index

We now introduce the algorithm, which builds a *ULISSE* index upon a data series collection. We maintain the structure of the $iSAX$ index [11], introduced in the preliminaries.

Each *ULISSE* internal node stores the Envelope $uENV$ that represents all the sequences in the subtree rooted at that node. Leaf nodes contain several Envelopes, which by construction have the same $iSAX(L)$. On the contrary, their $iSAX(U)$ varies, since it get updated with every new insertion in the node. (Note that, inserting by keeping the same $iSAX(U)$ and updating $iSAX(L)$ represents a symmetric and equivalent choice.)

In Figure 11, we show the structure of the *ULISSE* index during the insertion of an Envelope (rectangular/yellow box). Note that insertions are performed based on $iSAX(L)$ (underlined in the figure). Once we find a node with the same $iSAX(L) = (1-0-0-0)$ (Figure 11, $1^{st}step$) if this is an inner node, we descend its subtree (always following the $iSAX(L)$ representations) until we encounter a leaf. During this path traversal, we also update the $iSAX$ representation of the Envelope we are inserting, by increasing the number of bits of the segments, as necessary. In our example, when the Envelope arrives at the leaf, it has increased the cardinality of the second segment to two bits: $iSAX(L) = (1-\mathbf{10}-0-0)$, and similarly for $iSAX(U)$ (Figure 11, $2^{nd}step$). Along with the Envelope, we store in the leaf a pointer to the location on disk for the corresponding raw data series. We note that, during this operation, we do not move any raw data into the index.

To conclude the insertion operation, we also update the $iSAX(U)$ of the nodes visited along the path to the leaf, where the insertion took place. In our example, we update the upper part of the leaf Envelope to $iSAX(U) = (1-\mathbf{11}-0-0)$, as well as the upper part of the Envelope of the leaf's parent to $iSAX(U) = (1-\mathbf{1}-0-0)$ (Figure 11, $3^{rd}step$). This brings the *ULISSE* index to a consistent state after the insertion of the Envelope.

Algorithm 3 describes the procedure, which iterates over the series of the input collection C, and inserts them in the index. Note that function $bulkLoadingIndexing$ in Line 12 may use different bulk loading techniques. In our experiments, we used the iSAX 2.0 bulk loading algorithm [10]. Alongside the index, we also keep in memory (using the raw data order) all the Envelopes, represented by the symbols of the highest $iSAX$ cardinality available (Line 13). This information is used during query answering.

5.3.1 Space complexity analysis

The index space complexity is equivalent for the case of Z-normalized and non Z-normalized sequences. The choice of γ determines the number of $Envelopes$ generated and thus the index size. Hence, given a data series collection $C = \{D^1, ..., D^{|C|}\}$ the number of extracted Envelopes is given by $N = (\sum_i^{|C|} \lfloor \frac{|D^i|}{\ell_{min}+\gamma} \rfloor)$. If w PAA segments are used to discretize the series, each $iSAX$ symbol is represented by a single byte (binary label) and the disk pointer in each Envelope occupies b bytes (in general 8 bytes are used). The final space complexity is $O((2w)bN)$.

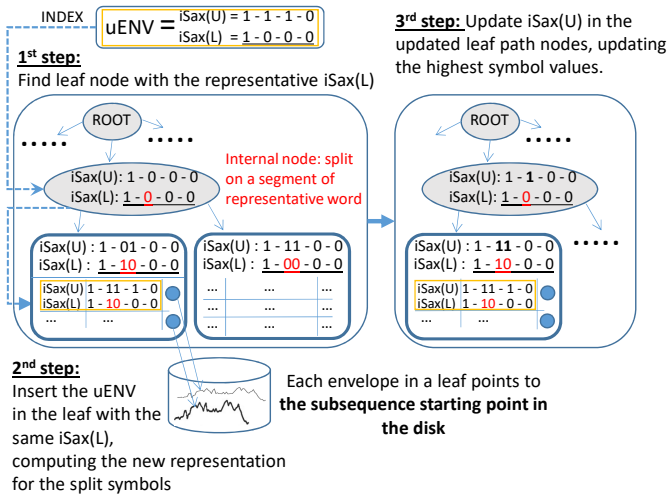


Fig. 11 Envelope insertion in an *ULISSE* index. $iSAX(L)$ is chosen to accommodate the Envelopes inside the nodes.

6 Similarity Search with ULISSE

In this section, we present the building blocks of the similarity search algorithms we developed for the *ULISSE* index, for both the Euclidean and the DTW distances, and both *k-NN* and ϵ-range queries.

We note that the same index structure supports both distance measures. When the query arrives, and depending on the distance measure we have chosen, we use the corresponding lower bounding and real distance formulas. We elaborate on these procedures in the following sections.

6.1 Lower Bounding Euclidean Distance

The iSAX representation allows the definition of a distance function, which lower bounds the true Euclidean [62]. This function compares the PAA coefficients of the first data series, against the iSAX breakpoints (values) that delimit the symbol regions of the second data series.

Let $\beta_u(S)$ and $\beta_l(S)$ be the breakpoints of the iSAX symbol S. We can compute the distance between a PAA coefficient and an iSAX region using:

$$distLB(PAA(D)_i, iSAX(D')_i) = \begin{cases} (\beta_u(iSAX(D')_i) - PAA(D)_i)^2 & if \beta_u(iSAX(D')_i) < PAA(D)_i \\ (\beta_l(iSAX(D')_i) - PAA(D)_i)^2 & if \beta_l(iSAX(D')_i) > PAA(D)_i \\ 0 & \text{otherwise.} \end{cases} \quad (3)$$

In turn, the lower bounding distance between two equi-length series D,D', represented by w PAA segments and w $iSAX$ symbols, respectively, is defined as:

$$mindist_{PAA\_iSAX}(PAA(D), iSAX(D')) = \sqrt{\frac{|D|}{w}} \sqrt{\sum_{i=1}^{w} distLB(PAA(D)_i, iSAX(D')_i)}. \quad (4)$$

We rely on the following proposition [36]:

Proposition 1 *Given two data series D, D', where $|D| = |D'|$, $mindist_{PAA\_iSAX}(PAA(D), iSAX(D')) \leq ED(D, D')$.*

Since our index contains Envelope representations, we need to adapt Equation 4, in order to lower bound the distances between a data series Q, which we call query, and a set of subsequences, whose iSAX symbols are described by the Envelope $uENV_{paaENV_{[D,\ell_{min},\ell_{max},a,\gamma,s]}} = [iSAX(L), iSAX(U)]$.

Therefore, given w, the number of PAA coefficients of Q, that are computed using the Envelope PAA segment length s on the longest multiple prefix, we define the following function:

$$mindist_{ULiSSE}(PAA(Q), uENV_{paaENV_{...}}) = \sqrt{s} \sqrt{\sum_{i=1}^{w} \begin{cases} (PAA(Q)_i - \beta_u(iSAX(U)_i))^2, & if (*) \\ (PAA(Q)_i - \beta_u(iSAX(L)_i))^2, & if (**) \\ 0 & \text{otherwise.} \end{cases}} \quad (5)$$

$$(*) \beta_u(iSAX(U)_i) < PAA(Q)_i$$

$$(**) \beta_l(iSAX(L)_i) > PAA(Q)_i$$

In Figure 12, we report an example of $mindist_{ULiSSE}$ computation between a query Q, represented by its PAA coefficients, and an Envelope in the iSAX space.

Proposition 2 *Given two data series Q,D, $mindist_{ULiSSE}(PAA(Q), uENV_{paaENV_{[D,\ell_{min},\ell_{max},a,\gamma,s]}}) \leq ED(Q, D_{i,|Q|})$, for each i such that $a \leq i \leq a + \gamma + 1$ and $|D| - (i-1) \geq \ell_{min}$.*

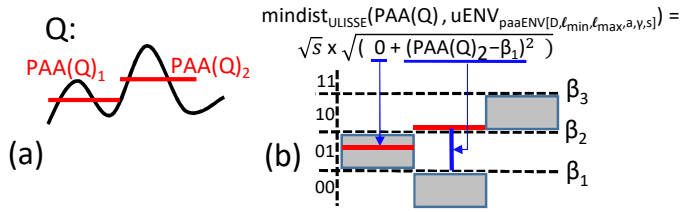


Fig. 12 Given the PAA representation of a query Q (*a*) and $uENV_{paaENV_{[D,\ell_{min},\ell_{max},a,\gamma,s]}}$ (*b*) we compute their $mindist_{ULiSSE}$. The $iSAX$ space is delimited with dashed lines and the relative breakpoints β_i.

Proof (sketch) We may have two cases, when $mindist_{ULiSSE}$ is equal to zero, the proposition clearly holds, since Euclidean distance is non negative. On the other hand, the function yields values greater than zero, if one of the first two branches is true. Let consider the first (the second is symmetric). If we denote with D'' the subsequence in D, such that $\beta_l(iSAX(U)_i) \leq PAA(D'')_i \leq \beta_u(iSAX(U)_i)$, we know that the upper breakpoint of the i^{th} iSAX symbol, of each subsequence in D, which is represented by the Envelope, must be less or equal than $\beta_u(iSAX(U)_i)$. It follows that, for this case, Equation 5 is equivalent to $distLB(PAA(Q)_i, iSAX(D'')_i)$, which yields the shortest lower bounding distance between the i^{th} segment of points in D and Q.

6.2 Lower Bounding Dynamic Time Warping

We present here a lower bound for the true DTW distance between two data series. Keogh et al. [30] introduced the LB_{Keogh} function, which provides a measure that is always smaller or equal than the true DTW, between two equi-length series. To compute this measure, we need to account for the valid warping alignments of two data series. Recall that the indexes of a valid path are confined by the Sakoa-Chiba band, where they are at most r points far from the diagonal (Euclidean Distance alignment). Given a data series D, we can build an envelope, $dtwENV_r(D)$, composed by two data series: L^{DTW} and U^{UDTW}, which delimit the space generated by the points of D that have indexes in the valid warping paths, constrained by the window r. Therefore, the i^{th} point of the two envelope sequences are computed as follows: $L_i^{DTW} = min(D_{(i-r,2r+1)})$ and $U_i^{DTW} = max(D_{(i-r,2r+1)})$. Intuitively, each i^{th} value of L^{DTW} and U^{DTW} represent the minimum and the maximum values, respectively, of the points in D that can be aligned with the i^{th} position of a matching series. In Figure 13(a), we report a data series D (plotted using a dashed line), contoured by its $dtwENV_r(D)$ envelope ($r = 7$).

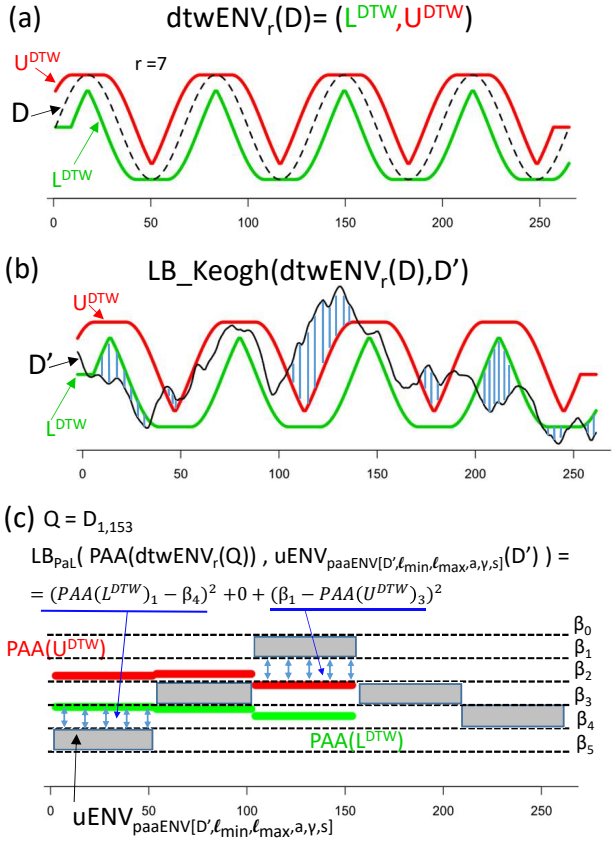


Fig. 13 *(a)* DTW Envelope (L^{DTW}, U^{DTW}) of a series D. *(b)* LB_{Keogh} distance between DTW Envelope and D'. *(c)* LB_{PaL} between the DTW Envelope of Q (prefix of D) and the *ULISSE* Envelope of D'. The horizontal dashed lines delimit the iSAX breakpoints space.

Lower bounding DTW. We can thus define the LB_{Keogh} distance [30], which is computed between a DTW envelope of a series D and a data series D', where $|D| = |D'|$ and the warping window is r:

$$LB_{Keogh}(dtwENV_r(D), D') = \sqrt{\sum_{i=1}^{|D|} \begin{cases} (D'_i - U_i^{DTW})^2, & if D'_i > U_i^{DTW} \\ (D'_i - L_i^{DTW})^2, & if D'_i < L_i^{DTW} \\ s0 & \text{otherwise.} \end{cases}} \quad (6)$$

The LB_{Keogh} distance between $dtwENV_r(D)$ and D' is guaranteed to be always smaller than, or equal to $DTW(D, D')$, computed with warping window r. In Figure 13(b), we depict the LB_{Keogh} distance between $dtwENV_r(D)$ (from Figure 13(a)), and a new series D'. The vertical (blue) lines represent the positive differences between D' and the DTW envelope of D, in Equation 6. Note that the computation of LB_{Keogh} takes $O(\ell)$ time (linear), whereas the true DTW computation runs in $O(\ell r)$ time using dynamic programming [30,57].

Lower bounding DTW in ULISSE. We now propose a new lower bounding measure for the true DTW distance between a data series and all the sequences (of the same length) represented by an *ULISSE* Envelope. To that extent, we first introduce a measure based on LB_{Keogh} distance, which is computed between the PAA representation of $dtwENV_r(D)$ and the $iSAX$ representation of D'. Given w, the number of PAA coefficients of each dtw envelope series (U^{DTW},L^{DTW}) that is equivalent to the number of iSAX coefficients of D', we have:

$$LB_{Keogh_{PAA.iSAX}}(PAA(dtwENV_r(D)), iSAX(D')) = \sqrt{\frac{|D|}{w}} \sqrt{\sum_{i=1}^{w} \begin{cases} (\beta_\ell(iSAX(D')_i) - PAA(U^{DTW})_i)^2, & if(*) \\ (PAA(L^{DTW})_i - \beta_u(iSAX(D')_i))^2, & if(**) \\ 0 & \text{otherwise.} \end{cases}}$$
$$(*)\beta_\ell(iSAX(D')_i) > PAA(U^{DTW})_i)$$
$$(**)PAA(L^{DTW})_i > \beta_u(iSAX(D')_i) \quad (7)$$

We know that $LB_{Keogh_{PAA.iSAX}}(PAA(dtwENV_r(D)), iSAX(D')) \leq LB_{Keogh}(dtwENV_r(D), D')$ as proven by Keogh et al. [30]. Given the PAA representation of $dtwENV_r(D)$ (of w coefficients), and an *ULISSE* Envelope built on D': $uENV_{paaENV_{[D',\ell_{min},\ell_{max},a,\gamma,s]}}) = [L, U]$, we define:

$$LB_{PaL}(PAA(dtwENV_r(D)), uENV_{paaENV[D',...]}) = \sqrt{s} \sqrt{\sum_{i=1}^{w} \begin{cases} (\beta_\ell(iSAX(L)_i) - PAA(U^{DTW})_i)^2, & if(*) \\ (PAA(L^{DTW})_i - \beta_u(iSAX(L)_i))^2, & if(**) \\ 0 & \text{otherwise.} \end{cases}}$$
$$(*)\beta_\ell(iSAX(L)_i) > PAA(U^{DTW})_i)$$
$$(**)PAA(L^{DTW})_i > \beta_u(iSAX(U)_i) \quad (8)$$

Lemma 3 *Given two data series D and D', where $\ell_{min} \leq |D| \leq \ell_{max}$, the distance $LB_{PaL}(PAA(dtwENV_r(D)), uENV_{paaENV[D',\ell_{min},\ell_{max},a,\gamma,s]})$ is always smaller or equal to $DTW(D, D'_{i,|D|})$, for each i such that $a \leq i \leq a + \gamma + 1$ and $|D'| - (i-1) \geq \ell_{min}$.*

Intuitively, the lemma states that the LB_{PaL} function always provides a measure that is smaller than the true DTW distance between D and each subsequence in D' of the same length, represented by $uENV_{paaENV[D',\ell_{min},\ell_{max},a,\gamma,s]})$.

Proof (sketch): We want to prove that

$$LB_{PaL}(PAA(dtwENV_r(D)), uENV_{paaENV[D',\ell_{min},\ell_{max},a,\gamma,s]})$$

is equal to

$$\underset{i}{\text{argmin}}\{LB_{Keogh_{PAA.iSAX}}(PAA(dtwENV_r(D)), iSAX(D'_{i,|D|}))\},$$

where $D'_{i,|D|}$ is a subsequence of D' represented by $uENV_{paaENV[D',\ell_{min},\ell_{max},a,\gamma,s]}$). The lemma clearly holds if LB_{PaL} yields zero, since the DTW distance between two series is always positive, or equal to zero. We thus test the case, where Equation 8 provides a strictly positive value. In the first case, the i^{th} lower iSAX breakpoint of L in the $ULISSE$ Envelope ($\beta_\ell(iSAX(L)_i)$) is greater than the i^{th} PAA coefficient of the U^{DTW}, namely $PAA(U^{DTW})_i$. This implies that any other i^{th} iSAX coefficient, which is contained in the *ULISSE* Envelope is necessarily greater than $\beta_\ell(iSAX(L)_i)$ and $PAA(U^{DTW})_i$. Hence, the Equation 8 is equivalent to the smallest value we can obtain from the first branch of $LB_{Keogh_{PAA.iSAX}}$ computed between each i^{th} iSAX coefficient of the subsequences in D' (represented in the $ULISSE$ Envelope) to the i^{th} PAA coefficient of $PAA(U^{DTW})$. $LB_{Keogh_{PAA.iSAX}}$ always yields a value that is smaller or equal to the true DTW distance, with warping window r.

The second case is symmetric. Here, the $\beta_u(iSAX(L)_i)$ coefficient is the closest to $PAA(L^{DTW})_i$, and greater than any other i^{th} iSAX coefficient of the $ULISSE$ Envelope. Therefore, Equation 8 is equivalent to the smallest value we can obtain on the second branch of $LB_{Keogh_{PAA.iSAX}}$ computed between each i^{th} iSAX coefficient of the subsequences in D' (represented in the $ULISSE$ Envelope) to the i^{th} coefficient of $PAA(L^{DTW})$. ■

In Figure 13(c), we depict an example that shows the computation of LB_{PaL} between the DTW Envelope that is built around the prefix of D (*153* points) and the *ULISSE* Envelope of the series D'. For this latter, the settings are: $a = 1$, $\ell_{min} = 153$, $\ell_{max} = 255$, $\gamma = 0$ and $s = 51$. In the figure, we represent the iSAX coefficients of the *ULISSE* Envelope, with (gray) rectangles delimited by their breakpoints (dashed horizontal lines). The coefficients of $PAA(U^{DTW})$ and $PAA(L^{DTW})$ are represented by red and green solid segments.

6.3 Approximate search

Similarity search performed on *ULISSE* index relies on Equation 5 (Euclidean distance) and Equation 8 (DTW distance) to prune the search space. This allows to navigate the tree, visiting the most promising nodes first. We thus provide a fast approximate search procedure we report in Algorithm 4. In Line 7 (or Line 9 if DTW distance is used), we start to push the internal nodes of the index in a priority queue, where the nodes are sorted according to their lower bounding distance to the query. Note that in the comparison, we use the largest prefix of the query, which is a multiple of the PAA segment length, used at the index building stage (Line 1). Recall that when the search is performed using the DTW measure, the PAA representation of the query is computed on the DTW envelope ($dtwENV_r$) of the segment-length multiple that completely contains the query (Line 2). This envelope is composed by two series, which encode the possible warping alignment according the warping window r. Therefore, the PAA representation is composed by two sets of coefficients, e.g., $PAA(L^{DTW})$ and $PAA(U^{DTW})$, as we depict in Figure 13.(c). Then, the algorithm pops the ordered nodes from the queue, visiting their children in the loop of Line 10. In this part, we still maintain the internal nodes ordered (Lines 34-35).

As soon as a leaf node is discovered (Line 12), we check if its lower bound distance to the query is shorter than the *bsf*. If this is verified, the dataset does not contain any data series that are closer than those already compared with the query. In this case, the approximate search result coincides with that of the exact search. Otherwise, we can load the raw data series pointed by the Envelopes in the leaf, which are in turn sorted according to their position, to avoid random disk reads. We visit a leaf only if it contains Envelopes that represent sequences of the same length as the query. Each time we compute either the true Euclidean distance (Line 19) or the true DTW distance (Line 21), the best-so-far distance (*bsf*) is updated, along with the R^a vector. Since priority is given to the most promising nodes, we can terminate our visit, when at the end of a leaf visit the k *bsf*'s have not improved (Line 22). Hence, the vector R^a contains the k approximate query answers.

6.4 Exact search

Note that the approximate search described above may not visit leaves that contain answers better than the approximate answers already identified, and therefore, it will fail to produce exact, correct results. We now describe an exact nearest neighbor search algorithm, which finds the k sequences with the absolute smallest distances to the query.

In the context of exact search, accessing disk-resident data following the lower bounding distances order may result in several leaf visits: this process can only stop after finding a node, whose lower bounding distance is greater than the *bsf*, guaranteeing the correctness of the results. This would penalize computational time, since performing many random disk I/O might unpredictably degenerate.

We may avoid such a bottleneck by sorting the Envelopes, and in turn the disk accesses. Moreover, we can exploit the *bsf* provided by approximate search, in order to perform a sequential search with pruning over the sorted Envelopes list (this list is stored across the *ULISSE* index). Intuitively, we rely on two aspects. First, the *bsf*, which can translate into a tight-enough bound for pruning the candidate answers. Second, since the list has no hierarchy structure, any Envelope is stored with the highest cardinality available,

Algorithm 4: *ULISSE* k-NN-*Approx*

```
   Input: int k, float [] Q, ULISSE index I, int r // warping window
   Output: float [k][|Q|] R^a, float [] bsf
1  float [] Q* ← PAA(Q_{1,..,⌊|Q|/I.s⌋});
2  float [][] Q^{*dtw} ← PAA(dtwENV_r(Q_{1,..,⌊|Q|/I.s⌋}));
3  float[k] bsf ← {∞, ..., ∞} ;
4  PriorityQueue nodes;
5  foreach node in I.root.children() do
6      if Euclidean distance search then
7          nodes.push(node, mindist_{ULiSSE}(Q*, node));
8      else if DTW search then
9          nodes.push(node, LB_{PaL}(Q^{*dtw}, node));
10 while n = nodes.pop() do
11     if n.isLeaf() and n.containsSize(|Q|) then
12         if n.lowerBound < bsf[k] then
               // sort according disk pos.
13             uENV [] Envelopes = sort(n.Envelopes);
               // iterate the Env. and compute true ED
14             oldBSF ← bsf[k];
15             foreach E in Envelopes do
16                 float [] D ← readSeriesFromDisk(E);
17                 for i ← E.a to min(E.a+E.γ+1,|D| − (|Q| − 1))
                       do
18                     if Euclidean distance search then
19                         ED_{updateBSF}(Q, E.D_{i,|Q|}, k,bsf, R^a);
20                     else if DTW search then
21                         DTW_{updateBSF}(Q, E.D_{i,|Q|}, k,bsf, R^a,r);
               // if bsf has not improved end visit.
22             if oldBSF == bsf[k] then
23                 break;
24         else
25             break; // Approximate search is exact.
26     else
27         LBleft ← 0, LBright ← 0;
28         if Euclidean distance search then
29             LBleft ←mindist_{ULISSE}(Q*, n.left);
30             LBright ←mindist_{ULISSE}(Q*, n.right);
31         else if DTW search then
32             LBleft←LB_{PaL}(Q^{*dtw}, n.left);
33             LBright←LB_{PaL}(Q^{*dtw}, n.right);
34         nodes.push(n.left, LBleft);
35         nodes.push(n.right, LBright);
```

Algorithm 5: *ULISSE* k-NN-*Exact*

```
   Input: int k, float [] Q, ULISSE index I, int r // warping window
   Output: float [k][|Q|] R
1  float [] Q* ← PAA(Q_{1,..,⌊|Q|/I.s⌋});
2  float [][] Q^{*dtw} ← PAA(dtwENV_r(Q_{1,..,⌊|Q|/I.s⌋}));
3  float [] bsf, float [k][|Q|] R ← k-NN-Approx(k, Q, I) ;
4  if bsf is not exact then
5      foreach E in I.inMemoryList do
6          LBDist← 0;
7          if Euclidean distance search then
8              LBDist←mindist_{ULiSSE}(Q*, E);
9          else if DTW search then
10             LBDist←LB_{PaL}(Q^{*dtw}, E);
11         if LBDist< bsf[k] then
12             float [] D ← readSeriesFromDisk(E);
13             for i ← E.a to min(E.a+E.γ+1,|D| − (|Q| − 1)) do
14                 if Euclidean distance search then
15                     ED_{updateBSF}(Q, D_{i,|Q|}, k,bsf, R);
16                 else if DTW search then
17                     l ← LB_{Keogh}(dtwENV_r(Q), D_{i,|Q|});
18                     if l <bsf[k] then
19                         DTW_{updateBSF}(Q, D_{i,|Q|}, k,bsf, R);
```

which guarantees a fine representation of the series, and can contribute to the pruning process.

Algorithm 5 describes the exact search procedure. In the case of Euclidean distance search, in Line 8 we compute the lower bound distance between the Envelope and the query. On the other hand, when DTW distance is used, we compute the lower bound distance in Line 10. If it is not smaller than the k^{th} *bsf*, we do not access the disk, pruning Euclidean Distance computations as well. Note that while we are computing the true distances, we can speed-up computations using the *Early Abandoning* technique [57], which works both for Euclidean and DTW distances. In the case of DTW distance, prior to computing the raw distance, we have a further possibility to prune computations using the LB_{Keogh} (Equation 6) in Line 17. This permits to obtain a lower bounding measure in linear time, avoiding the full DTW calculation.

6.5 Complexity of query answering

We provide now the time complexity analysis of query answering with *ULISSE*. Both the approximate and exact query answering time strictly depend on data distribution as shown in [74]. We focus on exact query answering, since approximate is part of it.

Best Case. In the best case, an exact query will visit one leaf at the stage of the approximate search (Algorithm 4), and during the second leaf visit will fulfill the stopping criterion (i.e., the *bsf* distance is smaller than the lower bounding distance between the second leaf and the query). Given the number of the first layer nodes (root children) N, the length of the first leaf path L, and the number of Envelopes in the leaf S, the best case complexity is given by the cost to iterate the first layer node and descend to the leaf keeping the nodes sorted in the heap: $O(w(N + LlogL))$, where w is the number of symbols checked at each lower bounding distance computation. We recall that computing the lower bound of Euclidean or DTW distance has equal time complexity. Moreover we need to take into account the additional cost of sorting the disk accesses and computing the true distances in the leaf, which is $O(S(logS + W))$ in the case of Euclidean distance, and $O(S(logS+rW))$ for DTW distance, where $W = \ell_{min}(\ell_{max} - \ell_{min} + \gamma + 1)$ represents the maximum number of points to check in each Envelope, and r is the warping window length. Note that we always perform disk accesses sequentially, avoiding random disk I/O. Each disk access in *ULISSE* reads at most $\Theta(\ell_{max} + \gamma)$ points.

Worst Case. The worst case for exact search takes place when at the approximate search stage, the complete set of leaves that we denote with T, need to be visited. This has a cost of $O(w(N + TLlogL))$ plus the cost of computing the true distances, which takes $O(T(S(logS + W)))$ for Euclidean distance, or $O(T(SlogS + SrW))$ for DTW distance, where (as above) $W = \ell_{min}(\ell_{max} - \ell_{min} + \gamma + 1)$. Note though that this worst case is pathological: for example, when all the series in the dataset are the same straight

lines (only slightly perturbed). Evidently, the very notion of indexing does not make sense in this case, where all the data series look the same. As we show in our experiments on several datasets, in practice, the approximate algorithm always visits a very small number of leaves.

***ULISSE* k-NN Exact complexity.** So far we have considered the exact k-NN search with regards to Algorithm 4 (approximate search). When this algorithm produces approximate answers, providing just an upper bound *bsf*, in order to compute exact answers we must run Algorithm 5 (exact search). The complexity of this procedure is given by the cost of iterating over the Envelopes and computing the *mindist*, which takes $O(Mw)$ time, where M is the total number of Envelopes in the index. Let's denote with V the number of Envelopes, for which the raw data are retrieved from disk and checked. Then, the algorithm takes an additional $O(VW)$ time to compute the true Euclidean distances, or $O(VrW)$ to compute the true DTW distances, with $W = \ell_{min}(\ell_{max} - \ell_{min} + \gamma + 1)$.

ϵ-range search adaption. We note that Algorithm 5 can be easily adapted to support ϵ-range search, without affecting its time complexity. In order to retrieve all answers with distance less than a given threshold $\epsilon \in \mathbb{R}$, we just need to replace the bound $bsf[k]$ with ϵ, in the test of line 11. Subsequently, if the test is true, the algorithm will compute the real distances between the query and all candidates in D (line 13), simply filtering the subsequences with distances lower than ϵ.

7 Experimental Evaluation

Setup. All the experiments presented in this section are completely reproducible: the code and datasets we used are available online [1]. We implemented all algorithms (indexing and query answering) in C (compiled with gcc 4.8.2). We ran experiments on an Intel Xeon E5-2403 (4 cores @ 1.9GHz), using the x86_64 GNU/Linux OS environment.

Algorithms. We compare *ULISSE* to *Compact Multi-Resolution Index (CMRI) [24]*, which is the current state-of-the-art index for similarity search with varying-length queries (recall that *CMRI* constructs a limited number of distinct indexes for series of different lengths). We note though, that in contrast to our approach, *CMRI* can only support non Z-normalized sequences. Furthermore, we compare *ULISSE* to *KV-Match* [67], which is the state-of-the-art indexing technique for ϵ-range queries that support the Euclidean and DTW measures over non Z-normalized sequences (remember that, as we discussed in Section 2, for Z-normalized data *KV-Match* only supports exact search for the constrained ϵ-range queries). Finally, we consider Index Interpolation *(IND-INT)* [43]. This method adapts an index based ϵ-range algorithm, which supports Z-normalization to answer k-NN queries of variable length.

In addition, we compare to the current state-of-the-art algorithms for subsequence similarity search, the *UCR suite* [57], and *MASS* [45]. Note that only *UCR suite* works with the Euclidean and DTW measures, whereas *MASS* supports only similarity search using Euclidean distance. These algorithms do not use an index, but are based on optimized serial scans, and are natural competitors, since they can process overlapping subsequences very fast.

Datasets. For the experiments, we used both synthetic and real data. We produced the synthetic datasets with a generator, where a random number is drawn from a Gaussian distribution $N(0, 1)$, then at each time point a new number is drawn from this distribution and added to the value of the last number. This kind of data generation has been extensively used in the past [74, 73], and has been shown to effectively model real-world financial data [16].

The real datasets we used are:

- (GAP), which contains the recording of the global active electric power in France for the period 2006-2008. This dataset is provided by EDF (main electricity supplier in France) [35];
- (CAP), the Cyclic Alternating Pattern dataset, which contains the EEG activity occurring during NREM sleep phase [64];
- (ECG) and (EMG) signals from Stress Recognition in Automobile Drivers [20];
- (ASTRO), which contains data series representing celestial objects [63];
- (SEISMIC), which contains seismic data series, collected from the IRIS Seismic Data Access repository [22].

In our experiments, we test queries of lengths *160-4096* points, since these cover at least 90% of the ranges explored in works about data series indexing in the last two decades [28, 5, 65].

7.1 Envelope Building

In the first set of experiments, we analyze the performance of the *ULISSE* indexing algorithm. Note that the indexing algorithm is oblivious to the distance measure used at query time.

In Figure 14(a) we report the indexing time (Envelope Building and Bulk loading operations) when varying γ. We use a dataset containing *5M* series of length *256*, fixing $\ell_{min} = 160$ and $\ell_{max} = 256$. Observe that, when $\gamma = 0$, the algorithm needs to extract as many Envelopes as the number of master series of length ℓ_{min}. This generates a significant overhead for the index building process (due to the maximal Envelopes generation), but also does not take into account the contiguous series of same length, in order to compute the statistics needed for Z-normalization. A larger γ speeds-up

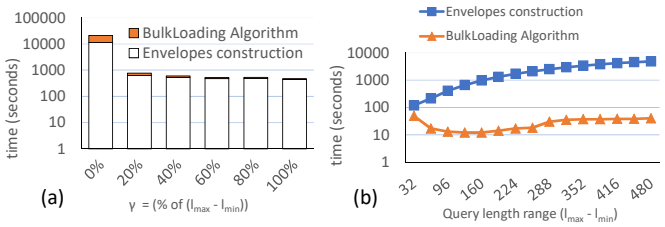
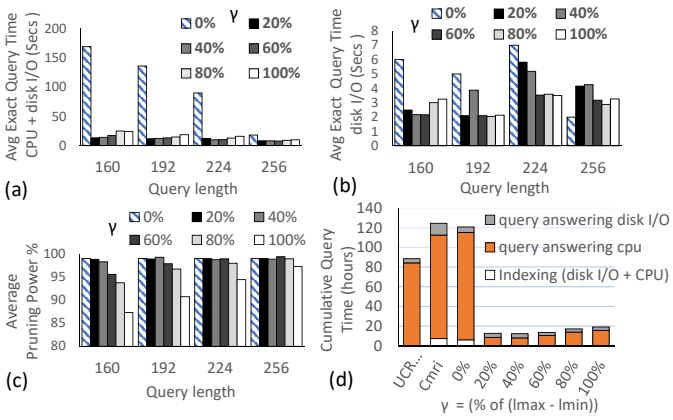


Fig. 14 (a) Construction and bulk Loading time (log scale) of Envelopes in 5GB datasets varying γ (5M of series of length 256), $\ell_{min} = 160$, $\ell_{max} = 256$. (b) Construction and Bulk Loading time (log scale) of Envelopes in 5GB dataset (2.5M of series of length 512) varying $\ell_{max} - \ell_{min}$ (lengths range), $\gamma = 256$, fixed $\ell_{max} = 512$.



(e)

Method	γ	Indexing time (h)	Indexing size (GB)	No. of Records/ Envelopes	No. of indexes
Cmri	-	7	90	2B	4
Ulisse	0%	6	58	485M	1
Ulisse	20%	0.2	3	25M	1
Ulisse	40%	0.11	1.7	15M	1
Ulisse	60%	0.08	1	10M	1
Ulisse	80%	0.08	1	10M	1
Ulisse	100%	0.04	0.5	5M	1

Fig. 15 Query answering time performance, varying γ on non Z-normalized data series. *a) ULISSE* average query time (CPU + disk I/O). *b) ULISSE* average query disk I/O time. *c) ULISSE* average query pruning power. *d)* Comparison of *ULISSE* to other techniques (cumulative indexing + query answering time). *e)* Table resuming the indexes' properties.

the Envelope building operation by several orders of magnitude, and this is true for a very wide range of γ values (Figure 14(a)). These results mean that the $uENV_{norm}$ building algorithm can achieve good performance in practice, despite its complexity that is quadratic on γ.

In Figure 14(b) we report an experiment, where γ is fixed, and the query length range ($\ell_{max} - \ell_{min}$) varies. We use a dataset, with the same size of the previous one, which contains *2.5M* series of length *512*. The results show that increasing the range has a linear impact on the final running time.

7.2 Exact Search Similarity Queries with Euclidean Distance

We now test *ULISSE* on exact 1-Nearest Neighbor queries using Euclidean distance. We have repeated this experiment varying the *ULISSE* parameters along predefined ranges, which are (default in bold) γ : $[0\%, 20\%, 40\%, 60\%, 80\%, \mathbf{100}\%]$, where the percentage is referring to its maximum value, ℓ_{min} : $[96, 128, \mathbf{160}, 192, 224, 256]$, ℓ_{max} : $[256]$, dataset series length (ℓ_S): $[\mathbf{256}, 512, 1024, 1536, 2048, 2560]$ and dataset size of $5GB$. Here, we use synthetic datasets containing random walk data in binary format, where a single point occupies 4 bytes. Hence, in each dataset C, where $|C|^{Bytes}$ denotes the corresponding size in bytes, we have a number of subsequences of length ℓ given by $N^{seq} = (\ell_S - \ell + 1) \times ((|C|^{Bytes}/4)/\ell_S)$. For instance, in a *5GB* dataset, containing series of length 256, we have $\sim$*500 Million* subsequences of length *160*.

We record the average *CPU time*, *query disk I/O time* (time to fetch data from disk: Total time - CPU time), and *pruning power* (percentage of the total number of Envelopes in the index that do not need to be read), of *100* queries, extracted from the datasets with the addition of Gaussian noise. For each index used, the *building time* and the relative *size* are reported. Note that we clear the main memory cache before answering each set of queries. We have conducted our experiments using datasets that are both smaller and larger than the main memory.

In all experiments, we report the cumulative running time of *1000* random queries for each query length.

Varying γ. We first present results for similarity search queries on *ULISSE* when we vary γ, ranging from *0* to its maximum value, i.e., $\ell_{max} - \ell_{min}$. In Figure 15, we report the results concerning non Z-normalized series (for which we can compare to *CMRI*). We observe that grouping contiguous and overlapping subsequences under the same summarization (Envelope) by increasing γ, affects positively the performance of index construction, as well as query answering (Figures 15(a) and (d)). The latter may seem counterintuitive, since γ influences in a negative way pruning power, as depicted in Figure 15(c). Indeed, inserting more master series into a single $Envelope$ is likely to generate large containment areas, which are not tight representations of the data series. On the other hand, it leads to an overall number of $Envelopes$ that is several orders of magnitude smaller than the one for $\gamma = 0\%$. In this last case, when $\gamma = 0$, the algorithm inserts in the index as many records as the number of master series present in the dataset (*485*M), as reported in (Figure 15(e)).

We note that the disk I/O time on compact indexes is not negatively affected at the same ratio of pruning power. On the contrary, in certain cases it becomes faster. For example, the results in Figure 15(b) show that for query length 160,

the $\gamma = 100\%$ index is more than 2x faster in disk I/O than the $\gamma = 0\%$ index, despite the fact that the latter index has an average pruning power that is 14% higher (Figure 15(c)). This behavior is favored by disk caching, which translates to a higher hit ratio for queries with slightly larger disk load. We note that we repeated this experiment several times, with different sets of queries that hit different disk locations, in order to verify this specific behavior. The results showed that this disk I/O trend always holds.

While disk I/O represents on average the $3 - 4\%$ of the total query cost, computational time significantly affects the query performance. Hence, a compact index, containing a smaller number of $Envelopes$, permits a fast in memory sequential scan, performed by Algorithm 5.

In Figure 15(d) we show the cumulative time performance (i.e., $4,000$ queries in total), comparing *ULISSE*, *CMRI*, and *UCR Suite*. Note that in this experiment, *ULISSE* indexing time is negligible w.r.t. the query answering time. *ULISSE*, outperforms both *UCR Suite* and *CMRI*, achieving a speed-up of up to $12x$.

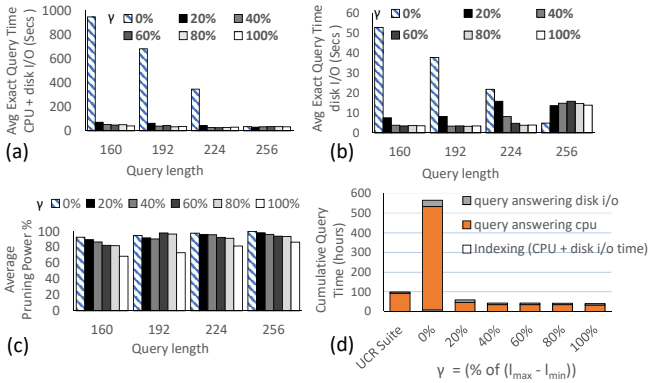
Further analyzing the performance of *CMRI*, we observe that it constructs four indexes (for four different lengths), generating more than $2B$ index records. Consequently, it is clear that the size of these indexes will negatively affect the performance of *CMRI*, even if it achieves reasonable pruning ratios.

These results suggest that the idea of generating multiple copies of an index for different lengths, is not a scalable solution.

In Figure 16, we show the results of the previous experiment, when using Z-normalization. We note that in this case the query answering time has an overhead generated by the Z-normalization that is performed on-the-fly, during the similarity search stage. Overall, we observe exactly the same trend as in non Z-normalized query answering. *ULISSE* is still $2x$ faster than the state-of-the-art, namely *UCR Suite*.

Varying Length of Data Series. In this part, we present the results concerning the query answering performance of *ULISSE* and *UCR Suite*, as we vary the length of the sequences in the indexed datasets, as well as the query length (refer to Figure 17). In this case, varying the data series length in the collection, leads to a search space growth, in terms of overlapping subsequences, as reported in Figure 17(e). This certainly penalizes index creation, due to the inflated number of Envelopes that need to be generated. On the other hand, *UCR Suite* takes advantage of the high overlapping of the subsequences during the in-memory scan. Note that we do not report the results for $CMRI$ in this experiment, since its index building time would take up to *1 day*. In the same amount of time, *ULISSE* answers more than $1,000$ queries.

Observe that in Figures 17(a) and (c), *ULISSE* shows better query performance than the UCR suite, growing linearly as the search space gets exponentially larger. This demonstrates that *ULISSE* offers a competitive advantage in terms of pruning the search space that eclipses the pruning techniques *UCR Suite*. The aggregated time for answering $4,000$ queries ($1,000$ for each query length) is 2x for *ULISSE* when compared to *UCR Suite* (Figures 17(b) and (d)).



(e)

γ	Indexing time (hours)	Indexing size (GB)	Number of Envelopes
0%	8,05	83,00	485M
20%	0,20	3,70	25M
40%	0,16	2,10	15M
60%	0,14	1,00	10M
80%	0,14	1,00	10M
100%	0,13	0,50	5M

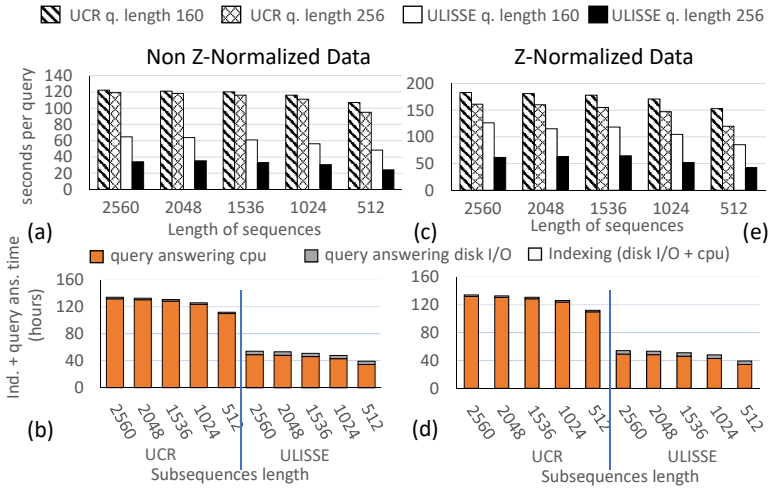
Fig. 16 *ULISSE* Indexing and exact queries performance on Z-normalized series, varying sigma. *a)* Average query time (CPU + disk I/O). *b)* Average query disk I/O time. *c)* Average query pruning power. *d)* Comparison to other techniques (cumulative indexing + query answering time). *e)* Table resuming the indexes' properties.

Comparison to Serial Scan Algorithms using Euclidean Distance. We now perform further comparisons to serial scan algorithms, namely, *MASS* and *UCR Suite*, with varying query lengths.

MASS [45] is a recent data series similarity search algorithm that computes the distances between a Z-normalized query of length l and all the Z-normalized overlapping subsequences of a single sequence of length $n \geq l$. *MASS* works by calculating the dot products between the query and n overlapping subsequences in frequency domain, in $logn$ time, which then permits to compute each Euclidean distance in constant time. Hence, the time complexity of *MASS* is $O(nlogn)$, and is independent of the data characteristics and the length of the query (l). In contrast, the *UCR Suite* effectiveness of pruning computations may be significantly affected by the data characteristics.

We compared *ULISSE* (using the default parameters), MASS and *UCR Suite* on a dataset containing *5M* data series of length *4096*. In Figure 17(f), we report the average query time (CPU + disk/io) of the three algorithms.

We note that MASS, which in some cases is outperformed by *UCR Suite* and *ULISSE*, is strongly penalized,



# subsequences in 5GB			Indexing Ulisse		
Length	160	256	Non Z-norm. time (secs)	Z-norm. time (secs)	#Envelopes
2560	1,200M	1,152M	377	2273	13,5M
2048	1,180M	1,120M	382	2187	13,75M
1536	1,147M	1,067M	392	2106	13,3M
1024	1,081M	961M	355	1977	13,75M
512	882M	642M	307	1445	12.5M

(e)

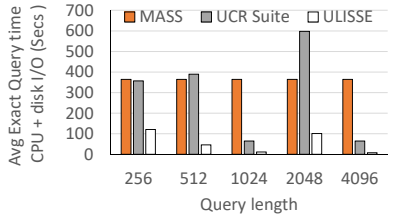


Fig. 17 Query answering time performance of *ULISSE* and *UCR Suite*, varying the data series size. Average query (CPU time + disk I/O) (*a*) for non Z-normalized, (*c*) for Z-normalized series). Cumulative indexing + query answering time (*b*) for non Z-normalized, (*d*) for Z-normalized series). *e)* Table resuming the indexes' properties. *f)* Comparison between MASS algorithm, *UCR Suite* and *ULISSE*.

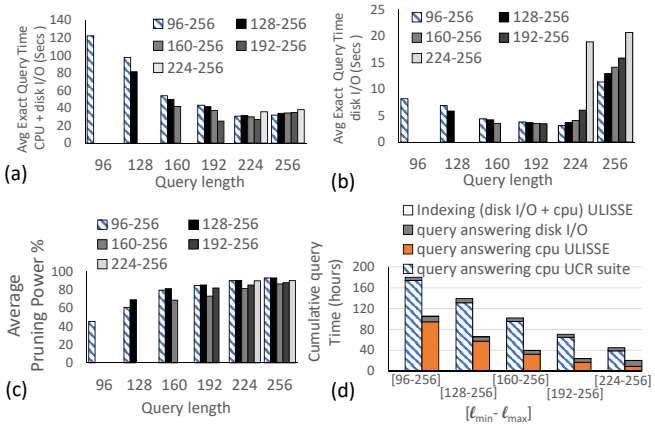


Fig. 18 Query answering time, varying the range of query length on Z-normalized data series. (*a) ULISSE* average query time (CPU + disk I/O). (*b) ULISSE* average query disk I/O time. (*c) ULISSE* average query pruning power. (*d) ULISSE* comparison to other techniques (cumulative indexing + query answering time).

when ran over a high number of non overlapping series. The reason is that, although MASS has a low time complexity of $O(nlogn)$, the Fourier transformations (computed on each subsequence) have a non negligible constant time factor that render the algorithm suitable for computations on very long series.

Varying Range of Query Lengths. In the last experiment of this subsection, we investigate how varying the length range $[\ell_{min}; \ell_{max}]$ affects query answering performance.

In Figure 18, we depict the results for Z-normalized sequences. We observe that enlarging the range of query length, influences the number of Envelopes we need to accommodate in our index. Moreover, a larger query length range corresponds to a higher number of Series (different normalizations), which the algorithms needs to consider for building a single Envelope (loop of line 16 of Algorithm 2). This leads to large containment areas and in turn, coarse data summarizations. In contrast, Figure 18(c) indicates that pruning power slightly improves as query length range increases. This is justified by the higher number of Envelopes generated, when the query length range gets larger. Hence, there is an increased probability to save disk accesses. In Figure 18(a) we show the average query time (CPU + disk I/O) on each index, observing that this latter is not significantly affected by the variations in the length range. The same is true when considering only the average query disk I/O time (Figure 18(b)), which accounts for $3 - 4\%$ of the total query cost. We note that the cost remains stable as the query range increases, when the query length varies between *96-192*. For queries of length *224* and *256*, when the range is the smallest possible the disk I/O time increases. This is due to the high pruning power, which translates into a higher rate of cache misses. In Figure 18(d), the aggregated time comparison shows *ULISSE* achieving an up to $2x$ speed-up over *UCR Suite*.

In Figure 19 we present the results for non Z-normalized sequences, where the same observations hold. Moreover, as we previously mentioned, when Z-normalization is not applied the pruning power slightly increases. This leads ULISSE to a performance up to $3x$ faster than *UCR Suite*.

7.3 Approximate Search Queries

Approximate Search with Euclidean Distance. In this part, we evaluate *ULISSE* approximate search. Since we compare our approach to CMRI, Z-normalization is not applied. Figure 20(a) depicts the cumulative query answering time for $4,000$ queries. As previously, we note that the in-

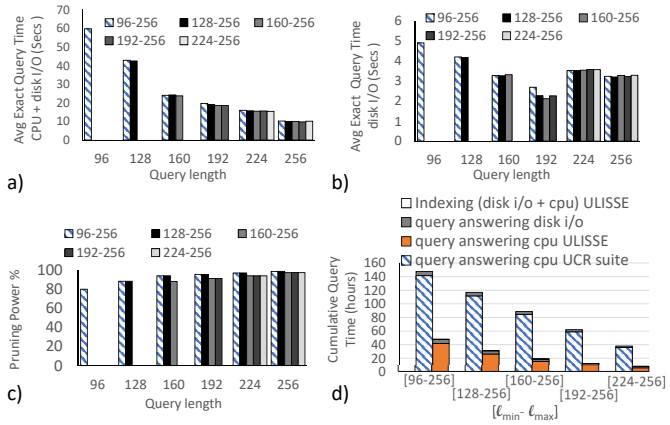
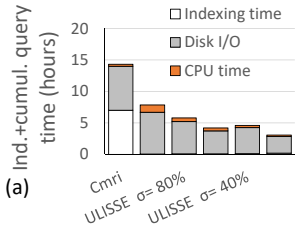


Fig. 19 Query answering time, varying the range of query length on non Z-normalized data series. *(a) ULISSE* average query time (CPU + disk I/O). *(b) ULISSE* average query disk I/O time. *(c) ULISSE* average query pruning power. *(d) ULISSE* comparison to other techniques (cumulative indexing + query answering time).



Query length	160	192	224	256
Ranking position	Ulisse			
1-25	92%	90%	92%	100%
26-50	8%	10%	8%	0%
	Cmri			
1-25	100%	100%	100%	100%

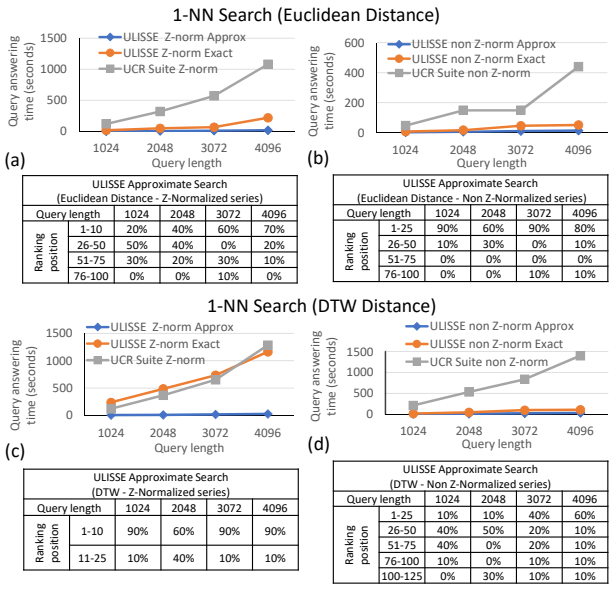
(b)

Fig. 20 Approximate query answering on non Z-normalized data series. *(a)* Cumulative Indexing + approximate search query time (CPU + disk I/O) of $4,000$ queries ($1,000$ per each query length in [160,192,224,256]). *(b)* Approximate quality: percentage of answers in the relative exact search range.

dexing time for *ULISSE* is relatively very small. On the other hand, the time that CMRI needs for indexing is 2x more than the time during which *ULISSE*s has finished indexing and answering $4,000$ queries.

In Figure 20(b), we measure the quality of the Approximate search. In order to do this, we consider the exact query results ranking, showing how the approximate answers are distributed along this rank, which represents the ground truth. We note that CMRI answers have better positions than the *ULISSE* ones. This happens thanks to the tighter representation generated by the complete sliding window extraction of each subsequence, employed by CMRI. Nevertheless, this small penalty in precision is balanced out by the considerable time performance gains: *ULISSE* is up to 15x faster than CMRI. When we use a smaller γ, (e.g., 20), *ULISSE* shows its best time performance. This is due to tighter $Envelopes$ containment area, which permits to find a better best-so-far with a shorter tree index visit.

Approximate Search with DTW. Here we evaluate, the time performance of query answering, along with the quality of approximate search. We test the search using both the Euclidean and DTW measures, on a synthetic series com-



ULISSE Approximate Search (Euclidean Distance - Z-Normalized series)

Query length	1024	2048	3072	4096
Ranking position 1-10	20%	40%	60%	70%
26-50	50%	40%	0%	20%
51-75	30%	20%	30%	10%
76-100	0%	0%	10%	0%

ULISSE Approximate Search (Euclidean Distance - Non Z-Normalized series)

Query length	1024	2048	3072	4096
Ranking position 1-25	90%	60%	90%	80%
26-50	10%	30%	0%	10%
51-75	0%	0%	0%	0%
76-100	0%	0%	10%	10%

ULISSE Approximate Search (DTW - Z-Normalized series)

Query length	1024	2048	3072	4096
Ranking position 1-10	90%	60%	90%	90%
11-25	10%	40%	10%	10%

ULISSE Approximate Search (DTW - Non Z-Normalized series)

Query length	1024	2048	3072	4096
Ranking position 1-25	10%	10%	40%	60%
26-50	40%	50%	20%	10%
51-75	40%	0%	20%	10%
76-100	10%	0%	10%	10%
100-125	0%	30%	10%	10%

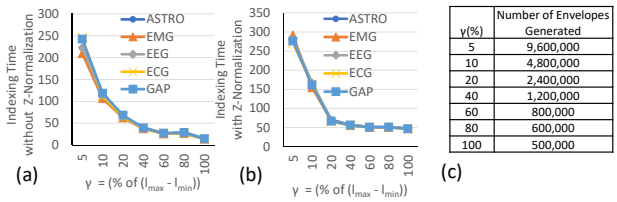
Fig. 21 Average query answering and approximate quality varying query length. *(a)* Z-normalized search with Euclidean distance. *(b)* Non Z-normalized search with Euclidean distance. *(c)* Z-normalized search with DTW measure. *(d)* Non Z-normalized search with DTW measure.

posed of $100M$ points. We test a query length range between $\ell_{min} = 1024$ and $\ell_{max} = 4096$. The other parameters are set to their default value.

In Figures 21(a) and (b), we report the average query answering time for the Z-normalized and non Z-normalized cases, respectively. The results show that *ULISSE* answers queries up to one order of magnitude faster than *UCR Suite*. Furthermore, we note that *ULISSE* scales better as the query length increases. This shows that our pruning strategy over summarized data, as well as having a good *bsf* approximate answer early on, represent a concrete advantage when pruning the search space.

In Figures 21(c) and (d), we report the time performance of query answering with the DTW measure, considering both Z-normalized and non Z-normalized search. In Figure 21(c), we observe that *ULISSE* answers queries slightly slower than *UCR Suite*, for three of the query lengths. This behavior is explained by the fact that the (overlapping) subsequences represented by the Envelopes have a total size $\sim$ 43x bigger than the original data points. In this case, the pruning power does not mitigate this disadvantage.

Overall, the results show that *ULISSE* is a scalable solution. Moreover, the approximate search, which in this experiment does not visit more than 5 leaves in the tree, represents a very fast solution, approximating well the exact answer (refer to the tables below each plot of Figure 21). We observed the same trend of visited leaves in all the experiments presented in this work. This means that in practice the time complexity of the Approximate search is very close to its best case having a constant query answering complexity.



(c)

γ(%)	Number of Envelopes Generated
5	9,600,000
10	4,800,000
20	2,400,000
40	1,200,000
60	800,000
80	600,000
100	500,000

Fig. 22 Indexing time of five real datasets (ASTRO, EMG, EEG, ECG, GAP) varying the number of master series in the Envelope (γ). The datasets contain $500K$ data series of length 256, whereas $\ell_{min} = 160$ and $\ell_{max} = 256$. *(a)* Indexing of non Z-Normalized series. *(b)* Indexing time of Z-Normalized series.

7.4 Experiments with Real Datasets

In this part, we discuss the results of indexing and query answering performed on real datasets. Here, we also consider the use of the Dynamic Time Warping (DTW) distance measure, along with Euclidean distance.

We start the evaluation by considering five different real datasets that fit the main memory. In the next sections, we will additionally consider real data series collections that do not fit in the available main memory. The objective of this experiment is to firstly assess the benefit of maximizing the number of subsequences represented by a *ULISSE* Envelope on query answering time. Moreover, we want to analyze the impact of the DTW measure on query time performance.

Indexing. For this experiment, we used five real dataset, where each one contains *500K* data series of length 256 (ASTRO, EMG, EEG, ECG, GAP). We show in Figure 22.(a,b) the indexing time performance, varying γ for both non Z-Normalized and Normalized sequences. Recall that γ is expressed as the percentage of the maximum number of master series that is $\ell_{max} - \ell_{min}$. The results confirm the trend depicted in Figure 14, where the time of building *ULISSE* Envelopes that contain all the master series of each series is one order of magnitude smaller than the time of building the most compact Envelopes, obtained with $\gamma = 5\%$. We also note that the overhead generated by the Z-normalization operations, which have an additional γ factor in the time complexity of the indexing algorithm, is amortized by the generation of $\sim$ 20x less Envelopes in the index, as depicted in Figure 22(c).

Query Answering with Euclidean Distance. We report in Figure 23 the results obtained for *1-NN* search over Z-normalized sequences, with Euclidean distance. All parameters are set to their default values. Therefore, in these experiments we used queries of length between $\ell_{min} = 160$ and $\ell_{max} = 256$; the series in the datasets have length 256. In Figure 23(a), we report the query pruning power as the number of master series (γ) in each Envelope varies. As expected, we can prune less candidates when the Envelopes contain more sequences. Recall that when a candidate (subsequence) is pruned, the search does not consider its raw values, thus avoiding both Z-normalization and Euclidean distance computations. If a candidate is not pruned, the search can abandon the computations earlier, when the running Euclidean Distance is greater than the k^{th} *bsf* distance.

In Figure 23(b), we report the average *abandoning power*, which measures the percentage of the total number of real Euclidean distance computations that are *not* performed. When the search processes an increased number of overlapping subsequences, we expect a decrease in the number of computations performed. We note that the search avoids computations when the Envelopes contain a large number of subsequences, namely, as γ increases.

In Figure 23(c), we report the average query time varying γ. We obtain the highest speed-up, with the most compact index (largest γ value), which is more that 2x faster than the state-of-the-art (*UCR Suite* algorithm). This confirms the trend we observed in the previous results conducted over synthetic data. We report the average query time for each dataset in Figure 23(d), and for each query length in Figure 23(e). In Figure 23(f), we show the average number of Euclidean distance and lower bound computations performed by *ULISSE* ($\gamma = 100\%$) and *UCR Suite*, as the query length varies (this corresponds to the average number of points on which the distance to the query is computed), as well as the number of points that are loaded from disk and Z-normalized (this corresponds to the overhead generated by the Z-normalization operations). The goal of this experiment is to quantify the overall benefit of *ULISSE* pruning and abandoning power. (Recall that *UCR Suite* does not perform any lower bound distance computations when using the Euclidean distance.)

First, we observe that *ULISSE* performs half of the Euclidean distance computations of *UCR Suite*, and considers up to seven time less points for the Z-normalization phase. Furthermore, we note that the computation of lower bound distances has a negligible impact on the query workload, especially when the query length is smaller than the length of the series in the dataset (256), in which case the number of candidate subsequences can be orders of magnitude more.

In Figure 24, we depict the results of query answering, without the use of Z-normalization. In this case, the results exhibit a small difference in terms of absolute pruning power values, which is higher when the search is performed on absolute series values. The average query answering time maintains the same trend we observe in Z-normalized query answering. On average, *ULISSE* has a 3x speed-up factor when compared to *UCR Suite*.

Query Answering with DTW Distance. We now report the results of query answering using the DTW measure (Figure 25). For this experiment, we used the default parameter settings, and the same real datasets considered in the previous two experiments. We study the efficiency of query an-

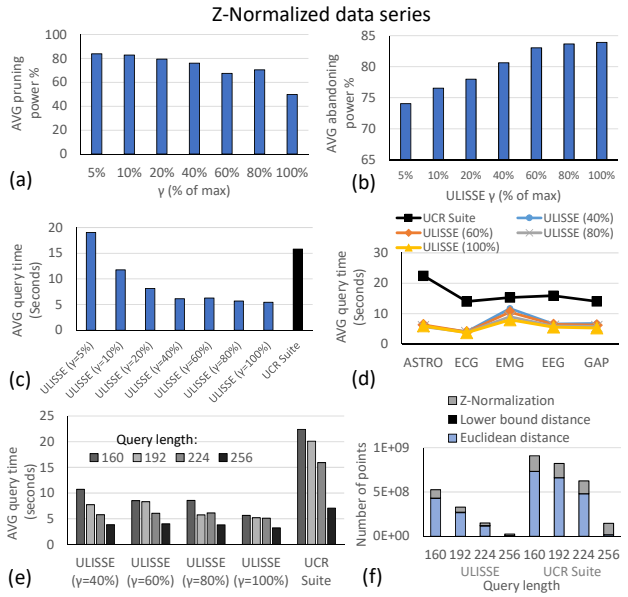


Fig. 23 Exact (Z-normalized) query answering with Euclidean distance on real datasets. *(a)* Average Pruning power *(b)* Average Abandoning power *(c)* Average query answering time *(d)* Average query answering time for each dataset *(e)* Average query answering time for each query length *(f)* Average query workload (number of points, with $\gamma = 100\%$)

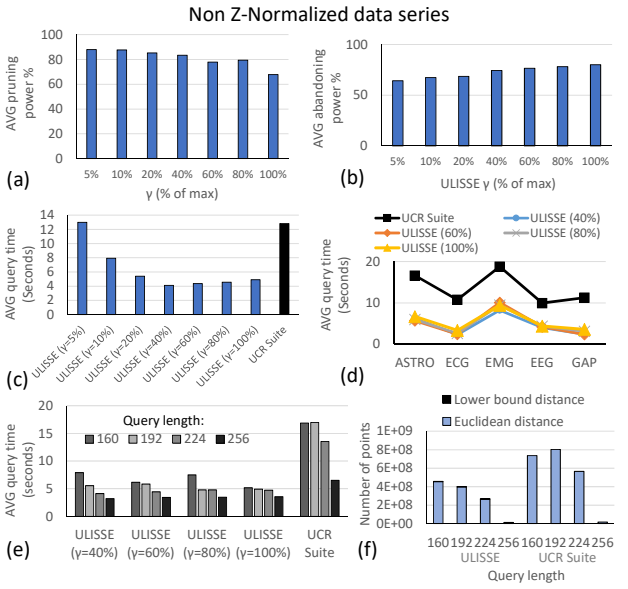


Fig. 24 Exact (non Z-normalized) query answering with Euclidean distance on real datasets. *(a)* Average Pruning power *(b)* Average Abandoning power *(c)* Average query answering time *(d)* Average query answering time for each dataset *(e)* Average query answering time for each query length *(f)* Average query workload (number of points, with $\gamma = 100\%$)

swering (*1-NN* query), which uses the DTW lower bounding measures to prune the search space.

In Figure 25(a) we report the average pruning power, when varying the DTW warping windows from 5% to 15% of the subsequence length. (These values for the warping window have commonly been used in the literature [30].) We vary γ between 60% and 100% of its maximum value, which give the best running time in this experiment. To avoid an unnecessary overload in the plot, we omit the results for γ smaller than 60%.

Once again, we note that the pruning power is negatively affected by the size of the Envelope (γ), and under DTW search the abandoning power slightly decreases as the $gamma$ and the $warping$ window get larger (see Figure 25(b)). This suggests that the DTW lower bound measure we propose is more sensitive than the one used for Euclidean Distance. Nevertheless, in the worst case *ULISSE* is still able to prune 20% of the candidates, and to abandon more than 80% of the DTW computations on raw values.

In Figure 25(c) we report the average query answering time varying γ, and in Figures 25(d) and (e) the average time for each dataset and for different query lengths, respectively, for $\gamma = 100\%$. For these last two experiments, we observe no significant difference for the other values of γ we tested.

We first note that, despite the loss of pruning power of ULISSE when increasing γ, the query answering time is not significantly affected (refer to Figures 25(c) and (e)). As in the case of Euclidean distance search, the compactness of the *ULISSE* index plays a fundamental role in determining the query time performance, along with the pruning and abandoning power.

In Figure 25(d), we note that only in the ECG and GAP datasets, enlarging the warping window has a substantial negative effect on query time ($2x$ slower), whereas in the other datasets, and in the worst case the time loss is equivalent to 10%.

In Figure 25(e), we report the average query workload of *ULISSE* and *UCR Suite*. In contrast to Euclidean distance queries, we notice that the largest amount of work corresponds to lower bounding distance computations. Recall that *ULISSE* prunes the search space in two stages: first comparing the query and the data in their summarized versions using LB_{PaL} (Equation 8), and then computing in linear time the LB_{Keogh} between the query and the non pruned candidates. In the worst case, the DTW distance point-wise computation are 10% of those performed for calculating the Lower Bound (query length 160). In general, the total number of points considered for the whole workload is up to 5x smaller than for *UCR Suite*. We note that the pruning strategy of *UCR Suite* is still very competitive, since it avoids a high number of true distance computations using the LB_{Keogh} lower bound. Nonetheless, it has to compute the lower bound distance on the entire set of candidates. The pruning strategy implemented in *ULISSE* permits to achieve up to $10x$ speedup over *UCR Suite*.

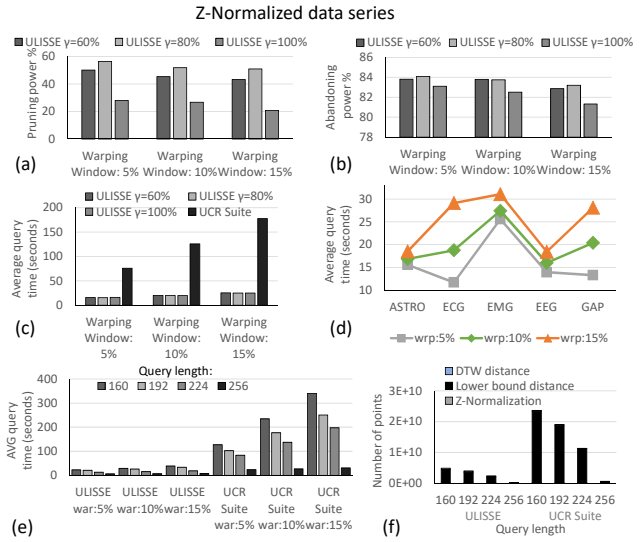


Fig. 25 Exact (Z-normalized) query answering with DTW measure on real datasets. *(a)* Average Pruning power (varying the warping window) *(b)* Average Abandoning power (varying the warping window) *(c)* Average query answering time *(d)* Average query answering time for each dataset *(e)* Average query answering time for each query length *(f)* Average query workload (number of points, with $\gamma = 100\%$)

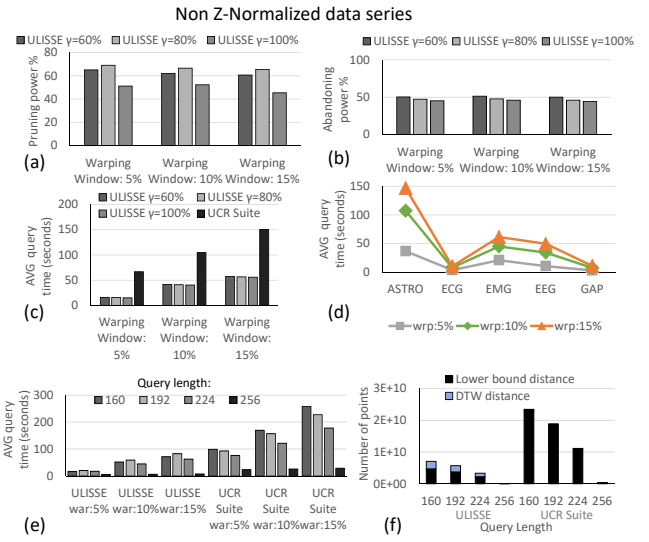


Fig. 26 Exact (Z-normalized) query answering with DTW measure on real datasets. *(a)* Average Pruning power (varying the warping window) *(b)* Average Abandoning power (varying the warping window) *(c)* Average query answering time *(d)* Average query answering time for each dataset *(e)* Average query answering time for each query length *(f)* Average query workload (number of points, with $\gamma = 100\%$)

In Figure 26, we report the results of DTW search, without the application of Z-Normalization. Also in this case, we note that the average pruning power of *ULISSE* is higher than the one we previously observed in the Z-normalized search (Figure 26(a)). On the other hand, the average abandoning power is less effective, as shown in Figure 26(b). As a consequence, we can see that the *ULISSE* search performs more DTW distance computations (refer to Figure 26(c)). Nevertheless, Figure 26(e) shows that on average *ULISSE* is up to 10x faster than *UCR Suite*, for all query lengths we tested.

Query over Large datasets with Euclidean Distance. Here, we test *ULISSE* on three large synthetic datasets of sizes *100*GB, *500*GB, and *750*GB, as well as on two real series collections, i.e., ASTRO and SEISMIC (described earlier). The other parameters are the default ones. For each generated index and for the *UCR Suite*, we ran a set of 100 queries, for which we report the average exact search time.

In Figure 27(a) we report the average query answering time (*1-NN*) on synthetic datasets, varying the query length. These results demonstrate that *ULISSE* scales better than *UCR Suite* across all query lengths, being up to 5x faster.

In Figure 27(b), we report the *k-NN* exact search time performance, varying k and picking the smallest query length, namely *160*. Note that, this is the largest search space we consider in these datasets, since each query has *9.7* billion of possible candidates (subsequences of length *160*). The experimental results on real datasets confirm the superiority of *ULISSE*, which scales with stable performance, also when increasing the number k of nearest neighbors. Once again it is up to 5x faster than *UCR Suite*, whose performance deteriorates as k gets larger.

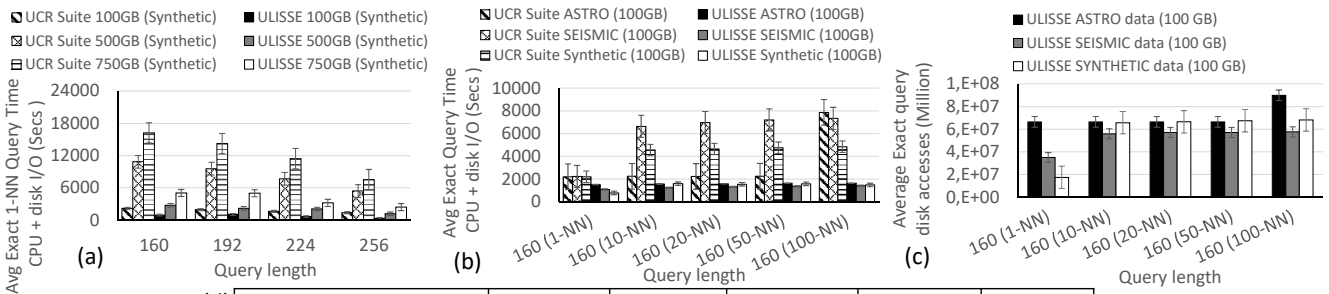
In Figure 27(c) we report the number of disk accesses of the queries considered in Figure 27(b). Here, we are counting the number of times that we follow a pointer from an envelope to the raw data on disk, during the sequential scan in Algorithm 5. Note that the number of disk accesses is bounded by the total number of Envelopes, which are reported in Figure 27(d) (along with the number of leaves and the building time for each index).

We observe that in the worst case, which takes place for the ASTRO dataset for $k = 100$, we retrieve from disk $\sim$*82%* of the total number of subsequences. This still guarantees a remarkable speed-up over *UCR Suite*, which needs to consider all the raw series.

Moreover, since *ULISSE* can use Early Abandoning during exact query answering, we observe during our empirical evaluation that disposing of the approximate answer distance prior the start of the exact search, permits to abandon on average *20%* of points more than *UCR Suite* for the same query.

Query over Large datasets with DTW. We conclude this part of the evaluation reporting the results of query answering on large datasets using the DTW distance.

In Figure 28, we report the time performance of (*1-NN* search) on the ASTRO, SEISMIC and synthetic datasets, each one containing $100M$ data series of length 256



(d)

	Synthetic data (100 GB)	Synthetic data (500 GB)	Synthetic data (750 GB)	Seismic data (100 GB)	Astro data (100 GB)
Indexing time (hours)	2.93	15.00	41.79	2.93	2.83
Leaf nodes in the ULISSE index	39,260	36,141	58,108	43,920	44,090
Envelopes in the ULISSE index	100M	453M	750M	100M	100M

Fig. 27 Exact and Approximate similarity search on Z-normalized synthetic and real datasets. *a)* Average exact query time (CPU + disk I/O) on synthetic datasets. *b)* Average exact *k-NN* query time (CPU + disk I/O) on real datasets (100 GB) varying k. *c)* Average disk accesses of *k-NN* query. *d)* Indexing measures for all datasets.

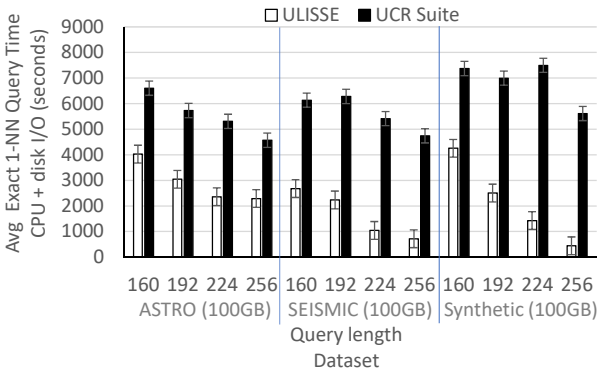


Fig. 28 Average exact query time with DTW distance (CPU + disk I/O) on real and synthetic datasets.

(100GB). Also in this case, *ULISSE* guarantees a consistent speed-up over *UCR Suite*, which is at least $\sim$ 1.5x faster in the worst case (ASTRO dataset, query length 160), and up to one order of magnitude faster (synthetic dataset, query length 256).

7.5 *ULISSE* vs Index Interpolation

In this section, we compare ULISSE to the Index Interpolation (*IND-INT*) method. *IND-INT* works by means of ϵ-range query answering of fixed length that serves the answering of variable length k-NN queries. We consider the real datasets previously introduced and each query is answered using $k = 1$ (Nearest Neighbor). We set the other parameters to their default values. The data series collections contain raw data series of fixed length 4096. Hence, the number of candidates changes according to the (variable) length of the query subsequence. We considered queries of lengths between 256-4096. In the left part of Figure 29(a), we report the total number of candidate answers for each of these lengths.

In order to test the scalability of the approaches as a function of the query length range, we test three different ranges: (256-512), (256-2048), and (256-4096). *IND-INT* must build an index using the smallest query length (256), extracting one record for each candidate. We thus have the same index for all three ranges. In contrast, for each one of the above three query length ranges, we can build a different *ULISSE* index, whose sizes are reported in the right part of Figure 29(a). Observe that all three *ULISSE* indexes have the same number of records (Envelopes), and that their sizes are two orders of magnitude smaller than the *IND-INT* index.

In Figure 29(b), we report the total query answering time (CPU + disk I/O; y-axis in log scale), as we vary the query length in the three chosen ranges. *ULISSE* answers 1-NN queries more than $10x$ faster than *IND-INT* in all these cases. We observe the same in Figure 29(c), where we report only the disk I/O time. Recall that *ULISSE* starts by performing an approximate search that visits first the most promising nodes of the index. In contrast, *IND-INT* issues an ϵ-range query, and thus must probe each summarized record in the index, and then access the disk, when the lower bounding distance between the query and the record is smaller than ϵ. This operation translates to significant time cost.

Note that in this set of experiments, we use for *IND-INT* an ϵ equal to the distance between the query and its approximate answer, which we obtain by first running the query using *ULISSE*. This method for choosing ϵ (proposed in the *RangeTopK* algorithm [19]) favors *IND-INT*, since it provides a good initial value.

Our experiments show that *ULISSE* scales better as the query length increases. As we observe in Figure 29(d), *ULISSE* always prunes more records as the query length increases, whereas *IND-INT* exhibits an unstable pruning pattern that depends on the query length.

a)

Original series length: 4096 – Number of series: 3134					
Subsequence length	# candidates	Subsequence length	# candidates	Subsequence length	# candidates
256	119M	**512**	111M	**1600**	77M
320	117M	**704**	105M	**2048**	63M
384	115M	**1152**	91M	**2176**	60M
448	113M	**1216**	89M	**3136**	30M

INDEX	#RECORDS (M)	INDEX SIZE (GB)
Interpolation (subsequence length 256)	119	20
Ulisse ℓ_{min}=256 - ℓ_{max}=512	1.2	0.27
Ulisse ℓ_{min}=256 - ℓ_{max}=2048	1.2	0.29
Ulisse ℓ_{min}=256 - ℓ_{max}=4096	1.2	0.32

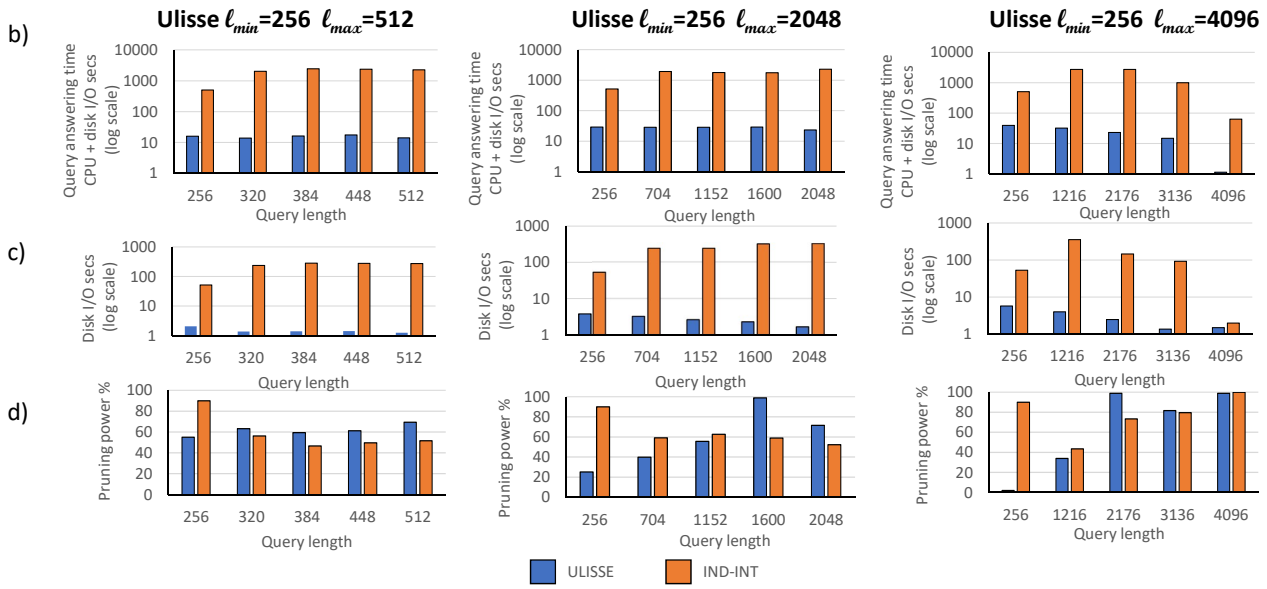


Fig. 29 Result of k-NN search comparison between *ULISSE* and *IND-INT*. (*a*) Data and Indices properties. (*b*) Query answering time. (*c*) Disk I/O. (*d*) Pruning power %.

Overall, *ULISSE* uses a more succinct index that permits to scale better (since there are less records to iterate over). This translates to reduced CPU time, as well as disk accesses.

7.6 ϵ-Range Queries

In this last part, we test the *ULISSE* search algorithm for the ϵ-Range query task. To that extent we adapted Algorithm 5, so that given as input $\epsilon \in \mathbb{R}$, it computes the set of subsequences that have a distance to the query smaller than or equal to ϵ. Similarly, we also adapted the *UCR Suite* to support ϵ-Range search. As additional competitors, we consider *IND-INT* (only for Euclidean distance), and *KV-Match*, which is the state-of-the art index-based solution for exact ϵ-Range queries on non Z-normalized data series.

In this experiment, we used five different real datasets, composed by a single data series of different lengths, as reported in Figure 30(a). For each of these datasets, we can see that *ULISSE* builds its index 5 times faster than *KV-Match*. This is because *KV-Match* is based on the construction of multiple indexes. Specifically, it builds different indexes performing a sliding windows extraction at different lengths. At query answering time, *KV-Match* performs a recombination of query answers coming from the different indexes.

For our ϵ-Range queries, we set the ϵ parameter to twice the *NN* distance of each query. In this manner, we simulate an exploratory analysis task. We report the average value of query selectivity in Figure 30(b). We note that in the ECG dataset the selectivity is very high. This is due to the periodic/cyclical nature of this kind of data, which contain repeating heartbeats subsequences that are very similar. In the other datasets, we have different values of selectivity ranging from 0.5% to 15%, when using Euclidean distance. On the other hand, when the DTW measure is considered, we observe a significant increase of the answer-set cardinality.

In Figures 30(c) and (d), we show the average query answering time for Euclidean distance, when varying the query length and the dataset, respectively. The results show that *IND-INT* does not represent a competitive alternative. In fact, only in one case, when the number of candidates is the smallest (i.e., 256), it has time performance better than the other approaches. When the number of possible answers increases (smaller query lengths), we observe that the performance of *IND-INT* becomes more than an order of magnitude worse than the rest.

We note that in this case $ULISSE$ and *KV-match* have no substantial difference in their time performance, with $ULISSE$ being slightly better.

However, when we consider the DTW distance, $ULISSE$ becomes up to one order of magnitude faster than

KV-Match, as shown in Figures 30(e) and (f). This difference is pronounced for the two largest datasets: $ULISSE$ is 3x faster for ECG, and 10x faster for GAP. It is important to note that since *KV-Match* needs to recombine the answers from the different index structures, its time performance is affected by this refinement phase of query answering, and is rather sensitive to dataset size and query selectivity.

8 Conclusions

Similarity search is one of the fundamental operations for several data series analysis tasks. Even though much effort has been dedicated to the development of indexing techniques that can speed up similarity search, all existing solutions are limited by the fact that they can only support queries of a fixed length.

In this work, we proposed *ULISSE*, the first index able to answer similarity search queries of variable-length, over both Z-normalized and non Z-normalized sequences, supporting the Euclidean and DTW distances, for answering exactly, or approximately both *k-NN* and ϵ-range queries. We experimentally evaluated, our indexing and similarity search algorithms, on synthetic and real datasets, demonstrating the effectiveness and efficiency (in space and time cost) of the proposed solution.

References

1. `http://www.mi.parisdescartes.fr/~mlinardi/ULISSE.html`.
2. Machine Learning in Time Series Databases (and Everything Is a Time Series !) Outline of Tutorial II. *Update*, pages 1–31.
3. Automatic detection of cyclic alternating pattern (cap) sequences in sleep: preliminary results. *Clinical Neurophysiology*, 1999.
4. I. Assent, R. Krieger, F. Afschari, and T. Seidl. The ts-tree: Efficient time series search and retrieval. In *EDBT*, 2008.
5. A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. J. Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *DAMI*, 2017.
6. A. J. Bagnall, R. L. Cole, T. Palpanas, and K. Zoumpatianos. Data series management. *Dagstuhl Reports*, 9(7), 2019.
7. P. Boniol, M. Linardi, F. Roncallo, and T. Palpanas. Automated Anomaly Detection in Large Sequences. In *ICDE*, 2020.
8. P. Boniol and T. Palpanas. Series2Graph: Graph-based Subsequence Anomaly Detection for Time Series. *PVLDB*, 2020.
9. Botao Peng (supervised by Panagiota Fatourou and Themis Palpanas). Data Series Indexing Gone Parallel. In *ICDE PhD Workshop*, 2020.
10. A. Camerra, T. Palpanas, J. Shieh, and E. J. Keogh. isax 2.0: Indexing and mining one billion time series. In *ICDM 2010*.
11. A. Camerra, J. Shieh, T. Palpanas, T. Rakthanmanon, and E. J. Keogh. Beyond one billion time series: indexing and mining very large time series collections with isax2+. *KAIS*, 2014.
12. M. Dallachiesa, T. Palpanas, and I. F. Ilyas. Top-k nearest neighbor search in uncertain data series. *PVLDB(8)1:13-24*, 2014.
13. K. Echihabi, K. Zoumpatianos, T. Palpanas, and H. Benbrahim. The lernaean hydra of data series similarity search: An experimental evaluation of the state of the art. *PVLDB*, 12(2), 2018.
14. K. Echihabi, K. Zoumpatianos, T. Palpanas, and H. Benbrahim. Return of the lernaean hydra: Experimental evaluation of data series approximate similarity search. *PVLDB*, 13(3), 2019.
15. ESA. SENTINEL-2 mission. `https://sentinel.esa.int/web/sentinel/missions/sentinel-2`.
16. C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *SIGMOD*, 1994.
17. A. Gogolou, T. Tsandilas, K. Echihabi, A. Bezerianos, and T. Palpanas. Data Series Progressive Similarity Search with Probabilistic Quality Guarantees. In *SIGMOD*, 2020.
18. A. Gogolou, T. Tsandilas, T. Palpanas, and A. Bezerianos. Progressive similarity search on time series data. In *BigVis, in conjunction with EDBT/ICDT*, 2019.
19. W. Han, J. Lee, Y. Moon, and H. Jiang. Ranked subsequence matching in time-series databases. In *VLDB*, 2007.
20. P. R. Healey JA. Detecting stress during real-world driving tasks using physiological sensors. *ITS 6(2):156-166*, June 2016.
21. P. Huijse, P. A. Estévez, P. Protopapas, J. C. Principe, and P. Zegers. Computational intelligence challenges and applications on large-scale astronomical time series databases. 2014.
22. IRIS. Seismic Data Access. `http://ds.iris.edu/data/access`, 2016.
23. F. Itakura. Minimum prediction residual principle applied to speech recognition. *IEEE Trans Acoust Speech Signal Process*, 1975.
24. S. Kadiyala and N. Shiri. A compact multi-resolution index for variable length queries in time series databases. *KAIS*, 2008.
25. T. Kahveci and A. Singh. Variable length queries for time series data. In *ICDE*, 2001.
26. K. Kashino, G. Smith, and H. Murase. Time-series active search for quick retrieval of audio and video. In *ICASSP*, 1999.
27. E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *KAIS*, 3, 2000.
28. E. J. Keogh and S. Kasetty. On the need for time series data mining benchmarks: A survey and empirical demonstration. *DAMI*, 2003.
29. E. J. Keogh, T. Palpanas, V. B. Zordan, D. Gunopulos, and M. Cardle. Indexing large human-motion databases. In *VLDB*, 2004.
30. E. J. Keogh and C. A. Ratanamahatana. Exact indexing of dynamic time warping. *Knowl. Inf. Syst.*, 2005.
31. H. Kondylakis, N. Dayan, K. Zoumpatianos, and T. Palpanas. Coconut: A scalable bottom-up approach for building data series indexes. *PVLDB (11)6:677-690*, 2018.
32. H. Kondylakis, N. Dayan, K. Zoumpatianos, and T. Palpanas. Coconut palm: Static and streaming data series exploration now in your palm. In *SIGMOD*, 2019.
33. H. Kondylakis, N. Dayan, K. Zoumpatianos, and T. Palpanas. Coconut: sortable summarizations for scalable indexes over static and streaming data series. *VLDBJ*, 28(6), 2019.
34. J. Kruskal and M. Liberman. The symmetric time-warping problem: From continuous to discrete. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, 01 1983.
35. M. Lichman. UCI machine learning repository, 2013.
36. J. Lin, E. Keogh, L. Wei, and S. Lonardi. Experiencing sax: a novel symbolic representation of time series. *DAMI*, 2007.
37. M. Linardi and T. Palpanas. ULISSE: ULtra compact Index for Variable-Length Similarity SEarch in Data Series. In *ICDE 2018*.
38. M. Linardi and T. Palpanas. Scalable, variable-length similarity search in data series: The ULISSE approach. *PVLDB*, 11(13):2236–2248, 2018.
39. M. Linardi, Y. Zhu, T. Palpanas, and E. J. Keogh. Matrix profile X: VALMOD - scalable discovery of variable-length motifs in data series. In *SIGMOD Conference 2018*.
40. M. Linardi, Y. Zhu, T. Palpanas, and E. J. Keogh. VALMOD: A suite for easy and exact detection of variable length motifs in data series. In *SIGMOD Conference 2018*.

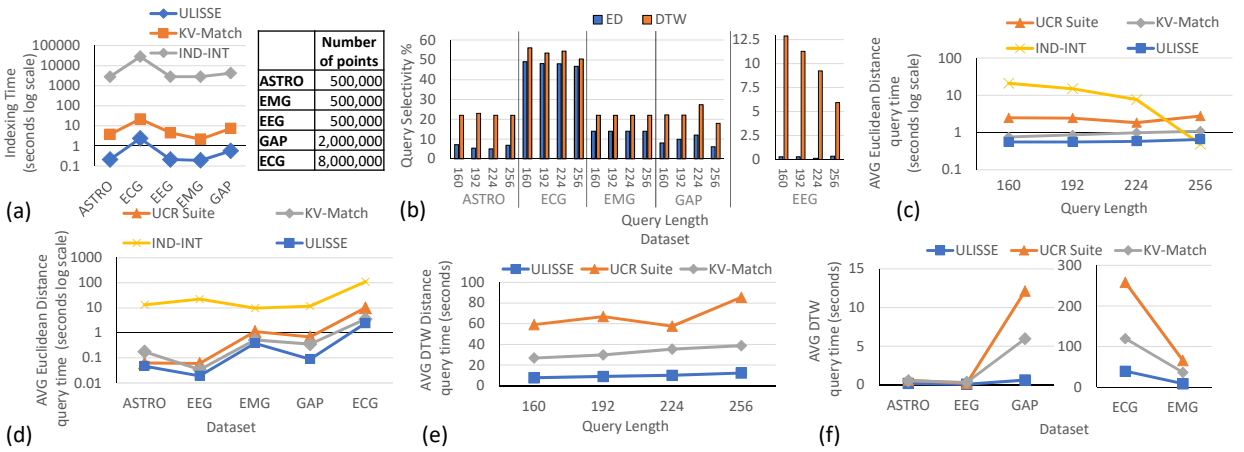


Fig. 30 Results of ϵ-range search on non Z-normalized real datasets. *(a)* Indexing time and datasets length. *(b)* Average selectivity of the queries in each dataset. *(c)* Average query answering time for each query length, using Euclidean distance. *(d)* Average query answering time for each dataset, using Euclidean distance. *(d)* Average query answering time for each query length, using DTW. *(e)* Average query answering time for each dataset, using DTW.

41. M. Linardi, Y. Zhu, T. Palpanas, and E. J. Keogh. Matrix Profile Goes MAD: Variable-Length Motif And Discord Discovery in Data Series. In *DAMI*, 2020.
42. J. Lines and A. Bagnall. Time series classification with ensembles of elastic distance measures. *DAMI*, 2015.
43. W. Loh, S. Kim, and K. Whang. A subsequence matching algorithm that supports normalization transform in time-series databases. *Data Min. Knowl. Discov.*, 2004.
44. Michele Linardi (supervised by Themis Palpanas). Effective and Efficient Variable-Length Data Series Analytics. In *VLDB PhD Workshop*, 2019.
45. A. Mueen, H. Hamooni, and T. Estrada. Time series join on subsequence correlation. In *ICDM*, 2014.
46. V. Niennattrakul and C. A. Ratanamahatana. On clustering multimedia time series data using k-means and dynamic time warping. MUE '07, 2007.
47. A. G. H. of Operational Intelligence Department Airbus. Personal communication., 2017.
48. T. Palpanas. Data series management: The road to big sequence analytics. *SIGMOD Rec.*, 2015.
49. T. Palpanas. Big sequence management: A glimpse of the past, the present, and the future. In *SOFSEM*, 2016.
50. T. Palpanas. The parallel and distributed future of data series mining. In *HPCS*, 2017.
51. T. Palpanas. Evolution of a Data Series Index - The iSAX Family of Data Series Indexes. *CCIS*, 2020.
52. T. Palpanas and V. Beckmann. Report on the First and Second Interdisciplinary Time Series Analysis Workshop (ITISA). *SIGMOD Rec.*, 48(3), 2019.
53. B. Peng, P. Fatourou, and T. Palpanas. Paris: The next destination for fast data series indexing and query answering. In *IEEE Big Data*, 2018.
54. B. Peng, T. Palpanas, and P. Fatourou. Messi: In-memory data series indexing. In *ICDE*, 2020.
55. B. Peng, T. Palpanas, and P. Fatourou. Paris+: Data series indexing on multi-core architectures. *TKDE*, 2020.
56. D. Rafiei and A. Mendelzon. Efficient retrieval of similar time sequences using dft. In *ICDE*, 1998.
57. T. Rakthanmanon, B. J. L. Campana, A. Mueen, G. E. A. P. A. Batista, M. B. Westover, Q. Zhu, J. Zakaria, and E. J. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *SIGKDD*, 2012.
58. U. Raza, A. Camerra, A. L. Murphy, T. Palpanas, and G. P. Picco. Practical data prediction for real-world wireless sensor networks. *IEEE Trans. Knowl. Data Eng.*, 2015.
59. H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Trans Acoust Speech Signal Process*, 1978.
60. P. Senin, J. Lin, X. Wang, T. Oates, S. Gandhi, A. P. Boedihardjo, C. Chen, and S. Frankenstein. Time series anomaly discovery with grammar-based compression. In *EDBT*, 2015.
61. D. Shasha. Tuning time series queries in finance: Case studies and recommendations. *IEEE Data Eng. Bull.*, 1999.
62. J. Shieh and E. J. Keogh. *i*sax: indexing and mining terabyte sized time series. In *KDD*, pages 623–631, 2008.
63. S. Soldi, V. Beckmann, W.H.Baumgartner, G.Ponti, C.R.Shrader, P. Lubinski, H.A.Krimm, F. Mattana, and J. Tueller. Long-term variability of agn at hard x-rays. *Astronomy & Astrophysics*, 2014.
64. M. G. Terzano, L. Parrino, A. Sherieri, R. Chervin, S. Chokroverty, C. Guilleminault, M. Hirshkowitz, M. Mahowald, H. Moldofsky, A. Rosa, R. Thomas, and A. Walters. Atlas, rules, and recording techniques for the scoring of cyclic alternating pattern (cap) in human sleep. *Sleep Medicine*, 2(6), 2001.
65. X. Wang, A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann, and E. J. Keogh. Experimental comparison of representation methods and distance measures for time series data. *DAMI*, 2013.
66. Y. Wang, P. Wang, J. Pei, W. Wang, and S. Huang. A data-adaptive and dynamic segmentation index for whole matching on time series. *PVLDB 6(10):793-804*, 2013.
67. J. Wu, P. Wang, N. Pan, C. Wang, W. Wang, and J. Wang. Kv-match: A subsequence matching approach supporting normalization and time warping. *ICDE*, 2019.
68. D. E. Yagoubi, R. Akbarinia, F. Masseglia, and T. Palpanas. Dpisax: Massively distributed partitioned isax. In *ICDM*, 2017.
69. D.-E. Yagoubi, R. Akbarinia, F. Masseglia, and T. Palpanas. Massively distributed time series indexing and querying. *TKDE*, 32(1), 2020.
70. K. Zoumpatianos, S. Idreos, and T. Palpanas. Indexing for interactive exploration of big data series. In *SIGMOD*, 2014.
71. K. Zoumpatianos, S. Idreos, and T. Palpanas. RINSE: interactive data series exploration with ADS+. *PVLDB (8)12*, 2015.
72. K. Zoumpatianos, S. Idreos, and T. Palpanas. ADS: the adaptive data series index. *VLDB J. 25(6): 843-866*, 2016.
73. K. Zoumpatianos, Y. Lou, I. Ileana, T. Palpanas, and J. Gehrke. Generating data series query workloads. *VLDB J.*, 27(6), 2018.
74. K. Zoumpatianos, Y. Lou, T. Palpanas, and J. Gehrke. Query workloads for data series indexes. In *SIGKDD*, 2015.
75. K. Zoumpatianos and T. Palpanas. Data series management: Fulfilling the need for big sequence analytics. In *ICDE*, 2018.