Efficient Interval-focused Similarity Search under Dynamic Time Warping

Jens Willkomm
Karlsruhe Institute of Technology (KIT)
jens.willkomm@kit.edu

Janek Bettinger
Karlsruhe Institute of Technology (KIT)
uuecs@student.kit.edu

Martin Schäler
Karlsruhe Institute of Technology (KIT)
martin.schaeler@kit.edu

Klemens Böhm
Karlsruhe Institute of Technology (KIT)
klemens.boehm@kit.edu

ABSTRACT

Similarity search on time series from large temporal text corpora is interesting in many settings. Our use case is the Google Books Ngram corpus and historians interested in the changes of word frequencies over time. More specifically, users are interested in similarity search in a specific period of time, aka. interval-focused similarity search. Related work formalizes interval-focused similarity search, but the sparsely existing approaches are limited to metric distance measures, like the Euclidean distance. Most other approaches in this area, that address the usage of warping distance measures, focus on whole matching similarity search. In this work, we present a novel search tree that uses so-called time series envelopes to group objects. To speed up the tree traversal, our search tree approximates the envelopes based on the node height, i. e., envelopes are tighter further down in the tree. We combine this with various time series pruning techniques, mainly to reduce the number of expensive distance computations. Our experimental evaluation shows that this combination is worthwhile and indeed decisive for a significant speedup, compared to less sophisticated adaptations of known approaches. We, first, show that a combination of both pruning groups of time series and single time series outperforms the usage of a single pruning technique. Secondly, we compare the wall-clock run times of our data structure to existing approaches and determine a significant speed up for focused-interval similarity search queries on large temporal data sets, like the Google Books Ngram corpus.

CCS CONCEPTS

• **Information systems** → **Nearest-neighbor search**.

KEYWORDS

data structure, time series, similarity search, dynamic time warping

ACM Reference Format:
Jens Willkomm, Janek Bettinger, Martin Schäler, and Klemens Böhm. 2019. Efficient Interval-focused Similarity Search under Dynamic Time Warping. In *16th International Symposium on Spatial and Temporal Databases (SSTD '19), August 19–21, 2019, Vienna, Austria.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3340964.3340969

SSTD '19, August 19–21, 2019, Vienna, Austria

ACM ISBN 978-1-4503-6280-1/19/08…$15.00
https://doi.org/10.1145/3340964.3340969

1 INTRODUCTION

Knowledge discovery features methods to support users from various disciplines to find information in large amounts of data. An example of users with a particular interest in temporal data are historians, e. g., to investigate the history of concepts or words. Our running example comes from this domain, even though our techniques are not limited to this field. A large real-world data set relevant to historians is the Google Books Ngram corpus [22]. It is a collection of words and their usage frequency in new books from a specific year. This frequency for every word is a time series over more than 200 years. Examples 1.1 and 1.2 describe information needs of historians regarding the Google Books Ngram corpus.

Example 1.1. Historians, conceptual historians in particular, study the origin of words and their meaning changes over time. They consider the usage frequencies of words over time as well as similarities of these frequencies [5, 25, 28]. For instance, to examine the social change coming and going with a dictatorship, a historian might question for words that rise and fall similarly to the word regime. *Since a social system needs some time to adjust to a change of the government form, interesting words might also rise and fall delayed to* regime. *Next, suppose that the interest of the historian is confined to the time interval between the two World Wars or the time interval from the end of World War 1 to the end of World War 2. In general, historians may want to look at various different time intervals in their studies.*

Example 1.1 illustrates the following general case: The information need is nearest neighbor search on time series confined to arbitrary intervals. This kind of information need (which has received little attention in the database literature so far) is called *interval-focused similarity search* [4]. Research has paid much more attention to kNN queries over the entire time interval [1, 10], which is called whole matching. Asfalg et al. [4] formalize the problem of interval-focused similarity search.

Example 1.2. A historian examines financial effects of the two World Wars. For this purpose, he searches for words similar to the word battleship *in the time interval from 1910 to 1960. When searching in the Google Books Ngram data, the result set contains the word* reparations, *whose shape is similar to the one of* battleship *with an offset of approximately 5 years. See Figure 1.*

Example 1.2 is an example of a meaningful kNN query on the Google Books Ngram corpus using dynamic time warping (DTW) as distance measure. The DTW distance measure allows warping, e. g., a temporal delay between the two time series.

In this work, we address the problem of efficiently answering interval-focused kNN queries on many time series. Here, a kNN search returns k time series with the smallest distance to the query

time series regarding a time series distance measure. Its generalized version, the interval-focused kNN search kNN($k, Q, [i, j]$), restricts the distance measure to a time range $[i, j]$ that the user specifies as part of the query. In this article, we focus on DTW as distance measure. However, this does not imposes any limitation, as one can easily replace the DTW measure with the Euclidean distance or any other L^p norm without violating any lower bound used subsequently. This is because these distances are special cases of DTW, i. e., the DTW distance without any alignment.

The literature gives only little attention to interval-focused similarity search. However, an extensive study — of whole matching similarity search — has resulted in two fundamental techniques. One common technique uses a spatial data structure, like an R- or R*-tree, to index time series using minimal bounding boxes [2, 3, 10, 11, 31]. This only works for metric distances, like the Euclidean distance, but does not work for semimetric distances that miss the triangle inequality, like the DTW distance [33]. In contrast, there are other techniques that use lower bounds for the DTW distance to prune either single time series [15, 16] or groups of time series [18, 24]. These techniques sequentially scan the data element-wise or group-wise. We refer to the first one as *lower bound* and the second one as *partitioning*. To sum up, most related work dos not support DTW as distance measure, considers every element, or does not support interval-focused queries. This indicates that efficiently answering interval-focused kNN queries is difficult.

To efficiently answer kNN queries on time series, like the one shown in Example 1.1, one must cope with the following challenges.

- *Handling Interval-focused Queries:* Since a query can refer to an arbitrary time interval, i. e., having an arbitrary start and end point, an index has to hold relevant information, like a lower bound estimation, for every possible time interval.
- *Efficient Pruning:* A data structure needs to organize the data in a way so that the search needs to only deal with a share of the elements. This reduces the number of distance computations.
- *High Dimensionality:* Time series are high dimensional objects. This typically leads to high tree-traversal costs.

In this article, we present the *Time Series Envelopes Index Tree* (TSEIT)[1], a novel tree-based index structure to efficiently evaluate interval-focused kNN queries. TSEIT is a dynamic data structure that supports operations to *insert*, *query* and *delete* time series. Our first contribution is the notion of hierarchically organized envelopes that form the basis of our tree structure TSEIT. The leaf nodes stores the time series, and the inner nodes stores envelopes that allow TSEIT to prune sub-trees during query evaluation. Our second contribution is a tree height based envelope approximation. It sets the envelope approximation degree based on the node's height. This speeds up the tree traversal near the root node and at the same time yields strong lower bounds near the leaf nodes. Our third contribution is a lower bound for arbitrary time intervals between an envelope and a time series. The lower bound holds for envelopes and time series in full dimensionality as well as for their piecewise aggregate approximation (PAA) representation. In Section 4, we describe the details of our data structure. We compare our approach against state-of-the-art methods for whole matching similarity search that we have modified to replace the whole time series with the queried interval during query time. Section 5 shows the superiority of our approach. TSEIT reduces the query execution time for a data set of 5 million time series by up to 50 %.

[1] TSEIT is pronounced [tsaɪt], like the German word *Zeit* for *time*

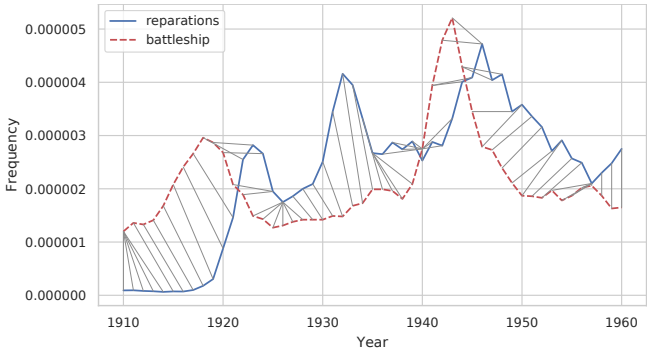

Figure 1: DTW's non-linear alignments between the usage frequency of the words "reparations" and "battleship" in the time interval [1910, 1960].

2 RELATED WORK

This section describes related work in the field of kNN search for time series. First, we discern interval-focused kNN search from other similarity search problems. Second, we present three ideas used in related work to accelerate time series similarity search. Finally, we explain our selection of reference points for the evaluation later in the paper.

2.1 Types of Time Series kNN Search

Categories of related work in the field of time series kNN search are *whole matching*, *subsequence matching* and *interval-focused matching* [4]. A whole matching query kNN(k, Q) returns the k time series with the smallest distance to the query time series Q over the full time period, i. e., the full time series length. There are various approaches for whole matching kNN search [6, 7, 15, 17, 24]. Whole matching is different from our problem, as one would have to, say, build a whole matching kNN index for each window $[i, j]$. A subsequence matching query kNN(ϵ, M) returns all time series containing a time series motif M, at any point in time. A time series motif is a pattern for time series [8]. Parameter ϵ defines the maximal deviation of the subsequence from the motif. Again, there are many approaches [9, 10, 12, 19, 21, 23, 27], and again, the problem is different from ours. An interval-focused matching query kNN($k, Q, [i, j]$) returns the k time series most similar to the query time series Q where only the similarity in the range $[i, j]$ matters [4]. This is the problem we address in this work.

2.2 Ideas to Accelerate Similarity Search

Due to the scarcity of work on interval-focused kNN, we have analyzed existing approaches from whole matching and subsequence matching and now review three ideas to accelerate time series kNN queries:

Spatial Access Methods A common technique to speed up similarity search for time series is to index time series with

a spatial search tree, e. g., an R-tree or one of its variants [1–3, 10–12, 15, 20, 31]. Vanilla spatial search trees are not ideal to handle time series for the following reasons. First, spatial search trees poorly handle high dimensional spaces [32, 34]. Second, the usage of bounding rectangles restricts an R-tree to metric distance measures [13], like the Euclidean distance. Third, spanning bounding rectangles across time series data points leads to hypercubes of vary large volumes and thus insufficiently separate the elements. To avoid the problem of high dimensional space indexing, some approaches extract time series features, e. g., wavelet coefficients, and index the feature vectors instead of the time series using a spatial tree [6, 7, 14, 16]. However, time series feature extraction approaches are unusable for interval-focused similarity search, since they usually remove the time domain.

Data Partitioning When partitioning similar time series into groups [24], a query starts with sequentially scanning the group most similar to the query. It continues scanning the groups in descending order of their similarity to the query and stops when the remaining groups are less similar than the best ones so far. This approach can achieve a high pruning rate if the time series groups have significantly different shapes. However, the number of partitions tends to grow linearly with the time series. This makes this approach inefficient for large data sets.

Lower Bounding Cascade Various approaches improve a sequential scan by checking a lower bound or a cascade of lower bounds before computing the real distance [26, 30]. This works best for long time series, as it saves time by pruning time series without computing its real distance. However, since it relies on a sequential scan, it has to consider every time series.

Based on this analysis, we conclude that any existing approach that can (also) be used for interval-focused kNN only uses *one* of the above ideas. Thus, we propose the first approach that hierarchically partitions the time series and uses lower bounds to speed up the scanning of candidate time series, and we study its performance.

2.3 Reference Points

In the experiments, our primary objective is to investigate whether one of the above ideas is dominant, i. e., the main reason for performance increases, or whether the combination of ideas is required. To this end, we rely on a sequential scan as baseline. Further, we use two state-of-the-art approaches: one applying only partitioning, named TWIST [24], and one using only lower bounds, named UCR Suite Cascading Lower Bounds [26]. As we describe in Section 4.4, our own approach generalizes both approaches, i. e., it can also be configured to mimic them.

3 BACKGROUND AND NOTATIONS

Based on the related work, this section describes some fundamentals and introduces our notation.

3.1 Time Series

A time series C is a sequence $\langle c_1, \ldots, c_i, \ldots, c_j, \ldots, c_n \rangle$ of length $n > 0$. $C[i, j]$ denotes a subsequence of time series C beginning at element c_i and ending at element c_j. S is a set of time series that contains $|S| = N$ time series. All time series of set S have the same length n. The Google Ngram corpus, our running example, has this property.

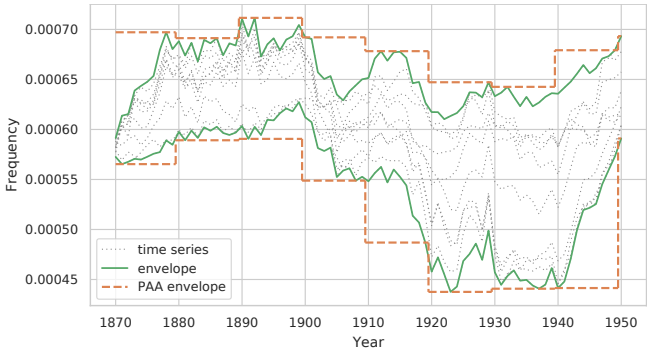

Figure 2: A set of time series enclosed by an envelope.

3.2 Interval-focused k-Nearest Neighbor

Querying for the k nearest neighbors of a query time series Q results in a set $R \subseteq S$ of $\min(k, |S|)$ time series so that for any two time series $C_R \in R$ and $C_S \in S \setminus R$ it holds that $d(Q, C_R) \leq d(Q, C_S)$ regarding a distance measure d. This kNN variant is whole-matching kNN. A generalization is the interval-focused kNN query, defined as $\text{kNN}(k, Q, [i, j])$ [4]. The interval-focused kNN query only considers the interval $[i, j]$ within time series Q and C to determine the distance. Obviously, $\text{kNN}(k, Q, [1, n]) = \text{kNN}(k, Q)$.

3.3 Dynamic Time Warping Distance

The result of a kNN query depends on the distance measure d. In this work, we focus on the dynamic time warping distance (DTW). Equation 2 shows the definition of distance $d(Q, C)$ between two time series Q and C of length m and n using DTW [29].

$$DTW(Q, C) = \sqrt[p]{D(m, n)} \tag{1}$$

$$D(i, j) = |q_i - c_j|^p + \min \begin{cases} D(i-1, j-1) \\ D(i-1, j) \\ D(i, j-1) \end{cases} \tag{2}$$

where $D(0, 0) = 0$ and $D(i, 0) = D(0, j) = \infty$ for $1 \leq i \leq m$ and $1 \leq j \leq n$. Figure 1 illustrates DTW's non-linear alignments for the usage frequencies of the words "reparations" and "battleship" in the Google Ngram corpus.

3.4 Lower Bound for a Group of Time Series

TSEIT organizes its time series in groups. This section says how we represent time series groups and how to compute a lower bound on a time series group. We describe (1) the idea of *envelopes* and define them, (2) the *segmentation* of envelopes, and (3) a lower bound distance for both an envelope and segmented envelope.

3.4.1 Time Series Envelope. An envelope $E = \langle e_1, \ldots, e_i, \ldots, e_n \rangle$ represents a set $\mathcal{C}$ of time series $C = \langle c_1, \ldots, c_i, \ldots, c_n \rangle$. Envelope E consists of elements $e_i = \langle ue_i, le_i \rangle$ representing a so-called

upper sequence $ue_i = \max_{C \in \mathcal{C}}(c_i)$ and a lower sequence $le_i = \min_{C \in \mathcal{C}}(c_i)$ [15]. Figure 2 illustrates the idea.

3.4.2 Piecewise Aggregate Approximation. A piecewise aggregate approximation (PAA) reduces the dimensionality of a time series, i. e., the number of data points [14]. To do so, PAA creates segments of a fixed size and aggregates all data points of a segment to one data point, e. g., to the mean, min, or max value. We refer to the PAA version of time series C with a fixed segment length of T as C^T. The DTW distance for two PAA time series C^T and Q^T is a lower bound for the DTW distance of the original time series C and Q [30]. Thus, $\text{DTW}(C^T, Q^T) \leq \text{DTW}(C, Q)$.

3.4.3 Envelope PAA. Since an envelope consists of an upper and a lower sequence, one can create a PAA version E^T of an envelope E. To receive a valid PAA envelope, one must aggregate the upper sequence to the maximum and the lower sequence to the minimum [24]. Equation 3 defines a PAA envelope.

$$E^T = \langle e_1^T, \ldots, e_i^T, \ldots, e_t^T \rangle \tag{3}$$

$$e_i^T = \langle ue_i^T, le_i^T \rangle \tag{4}$$

$$ue_i^T = \max(e_x, \ldots, e_y) \tag{5}$$

$$le_i^T = \min(e_x, \ldots, e_y) \tag{6}$$

where $x = (i-1) \cdot T + 1$ is the start of a segment and $y = i \cdot T$ the end. Figure 2 illustrates an envelope and its coarse PAA variant.

3.4.4 Group Lower Bound. The lower bound for a group of time series (LBG) is a lower bound of the DTW distance from a time series Q to an envelope E, i. e., to all time series that envelope E encloses [24]. It holds that $\text{LBG}(Q, E) \leq \arg\min_{C \in \mathcal{C}} \text{DTW}(Q, C)$. LBG works as follows. For every point in time i, the data point q_i of time series Q can either be inside envelope E, i. e., $le_i \leq q_i \leq ue_i$, or outside of it, i. e., $q_i > ue_i$ or $q_i < le_i$. If data point q_i is outside the envelope, LBG adds the DTW distance from q_i to the nearest envelope border, i. e., either ue_i or le_i. In turn, if data point q_i is inside the envelope, LBG adds a distance of 0 for point i. If envelope E encloses a time series Q, a lower bound is 0. A proof is in [24].

LBG is also a valid lower bound on the PAA representations Q^T and E^T of a time series Q and an envelope E [30]. It holds that $\text{LBG}(Q^T, E^T) \leq \text{LBG}(Q, E)$. This may lead to a less tight lower bound, but may speed up its calculation of the lower bound significantly.

Equation 7 shows the definition of the LBG lower bound.

$$\text{LBG}(Q^T, E^T) = \sqrt[p]{D(m, n)} \tag{7}$$

$$D(i,j) = T \cdot D_{seg}(q_i^T, e_j^T) + \min \begin{cases} D(i-1, j-1) \\ D(i-1, j) \\ D(i, j-1) \end{cases} \tag{8}$$

$$D_{seg}(q_i^T, e_j^T) = \begin{cases} |lq_i^T - ue_j^T|^p & \text{if } lq_i^T > ue_j^T \\ |le_j^T - uq_i^T|^p & \text{if } le_j^T > uq_i^T \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

where $D(0,0) = 0$ and $D(i,0) = D(0,j) = \infty$ for $1 \leq i \leq m$ and $1 \leq j \leq n$.

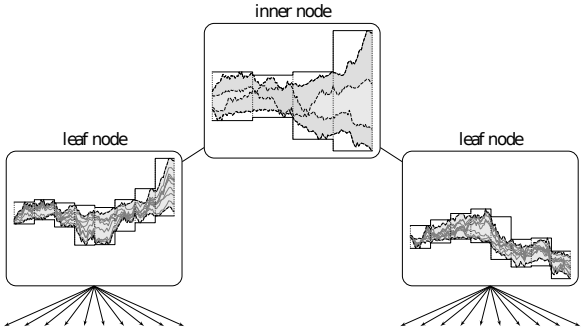

Figure 3: An example of a Time Series Envelopes Index Tree with two leaf nodes and a single inner node.

Observe the following properties of LBG. First, a smaller segment size T usually leads to a tighter lower bound. Second, Equation 7 is also valid for the LBG calculation on the full dimensionality, i. e., segment size $T = 1$. Third, LBG is also a valid lower bound for the DTW distance of two time series [30]. In this case, envelope E only contains a single time series element. On a segment size of $T = 1$, LBG leads to the exact DTW distance of two time series. Fourth, LBG also works for different segment sizes for Q^{T_1} and E^{T_2} when substituting factor T with $min(T_1, T_2)$ in Equation 8 [30]. We will use this property to implement interval-focused queries.

3.5 Cascading Lower Bounds

Rakthanmanon et al. analyze and compare the runtimes and tightness of lower bounds for the DTW distance [26]. To achieve a good compromise between run time and pruning factor, they propose to use two lower bounds in a cascade, as follows.

- **LB_KimFL(Q, C)** was published by Kim et at. [16] and modified by Rakthanmanon et al. [26]. It is the Euclidean distance for the first (F) and last (L) point of time series Q and C, since these points always have an alignment of 0. Kim's lower bound has a run-time complexity of $O(1)$. Thus, it is a very fast lower bound that removes many time series in spite of its looseness.
- **LB_Keogh(Q, C)** was published by Keogh et al. [15]. It is the Euclidean distance between a bounding envelope around time series Q and time series C. Its run-time complexity is in $O(n)$, i. e., depends linearly on the length of the time series. Thus, Keogh's lower bound is cheaper than the quadratic run-time of DTW, but prunes further candidate time series.

Rakthanmanon et al. use $max(LB\_Keogh(Q, C), LB\_Keogh(C, Q))$ as third step in their lower bound cascade [26], since LB_Keogh is not commutative, i. e., $LB\_Keogh(Q, C) \neq LB\_Keogh(C, Q)$.

4 OUR TSEIT APPROACH

In this section, we introduce our novel Time Series Envelopes Index Tree (TSEIT) data structure. First, we focus on the tree structure and, second, describe the operations.

Algorithm 1: query(Q, k, [i, j])

```
Input: Q ← query time series
k ← desired number of results
[i, j] ← query interval
Data: root ← root node of TSEIT
L ← a priority queue
R ← a sorted list of tuples of ⟨DTW(Q, C), C⟩
bsf ← the best-so-far distance
Result: A list with k elements having the lowest DTW distance to time
        series Q
1  begin
2      L.enqueue(root)
3      while L.has_next() do
4          node ← L.next()
5          T ← 2^node.height                    /* node's segment size */
6          if bsf ≤ LBG(Q^T[i, j], node.E^T[i, j]) then
7              break
8          if node.is_leaf_node() then
9              cands ← seq_scan(node, Q[i, j], k, bsf)
10             R.insert(cands); R.set_size(k)
11             bsf ← max(R.get_distances())
12         else
13             L.enqueue(node.get_child_nodes())
14     return R
```

4.1 Tree Structure

TSEIT is a search tree that stores time series envelopes in its nodes. Envelopes seem to be a more suitable representation of time series groups than rectangles. Figure 3 illustrates a Time Series Envelopes Index Tree with two leaf nodes and one inner node. TSEIT is a balanced tree that grows upwards, i. e., toards the root. Every node stores an envelope that wraps all time series reachable by this node. TSEIT distinguishes between two types of nodes: leaf nodes and inner nodes including the root node. Leaf nodes hold envelopes in full dimensionality, whereas inner nodes store PAA envelopes. The envelopes of the inner nodes towards the root grow into *two* directions: (1) They grow on the "domain axisW: The envelope encloses the minimum and maximum of its child nodes. (2) They grow on the time axis, i. e., the PAA segment size increases with the tree height and thus the envelope dimensionality of higher nodes decreases. In other words, we start from the root node with a loose envelope that gets tighter every step downwards the tree on the domain axis as well as on the time axis. Using envelopes instead of bounding boxes is our solution to address the problem of large-volume rectangles discussed in Section 2. Figure 3 illustrates TSEIT's envelope-growing behavior. We describe the PAA envelope of inner nodes in the following.

4.1.1 PAA Envelope of Inner Node. Recall that leaf nodes store envelopes in full dimensionality. This is equal to a PAA representation with a segment size of 1. In contrast, inner nodes at the same height have the same dimensionality. Parent nodes double the PAA segment size of their children. This is, inner nodes at height 1 have envelopes with segment size 2, while the nodes one level up have envelopes with segment size 4 and so on. Since TSEIT grows upwards and thus all its child nodes have the same depth, the PAA segment size T of a node is a function of the node's height h: $T(h) = 2^h$.

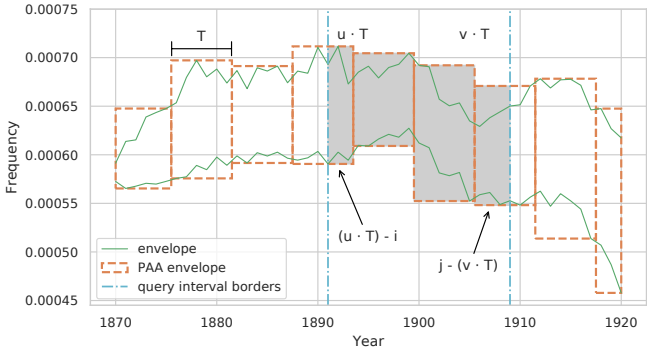

Figure 4: A LBG computation that partially includes the PAA segments at start and end of interval [*i*, *j*].

4.2 Interval-focused Query

To answer query($Q, k, [i, j]$), TSEIT searches its leaf nodes in ascending order of the node's lower bound to Q until the lower bound distance is larger than the best-so-far distance. See Algorithm 1.

The following describes specifics of TSEIT's query algorithm: (1) tree traversal with PAA envelopes, (2) query an interval that differs from PAA segment borders, and (3) scanning a leaf node.

4.2.1 Tree Traversal Based on PAA Envelopes. The tree traversal is based on the LBG lower bound between query time series Q^T and the node's envelope E^T (Line 6). Q^T and E^T are PAA versions of the query time series and the envelope of the current node. Line 5 specifies the PAA segment length used on the current tree level. Note that the PAA version of the Envelope E^T is already available. Note also that we need to compute the PAA representations of the query time series Q^T only once per query. The number of necessary PAA representations depends on the height of the tree. Using height based PAA representations has the following nice properties: It speeds up the tree traversal, and the lower bounds become more accurate towards the leaf nodes.

4.2.2 Query Interval Border Inside a PAA Segment. The approximation with PAA replaces all data points inside a segment with a single data point. The query intervals i and j might lie inside a PAA segment. We call these segments *side segments*. Figure 4 illustrates a query whose interval borders lie inside a PAA segment. Including the side segments to the lower bound invalidates the bound. Excluding the side segments leads to a less tight lower bound and thus to a lower pruning rate. Therefore, we modify the lower bound to partially include the side segments and thus keep the lower bound's tightness in arbitrary intervals. Inside the interval, the first PAA border $u = \arg\min_{1 \le w \le t}(w \cdot T \ge i)$ starts at position $u \cdot T$, and the last PAA border $v = \arg\max_{1 \le w \le t}(w \cdot T \le j)$ starts at position $v \cdot T$. We have to include segment u with size $(u \cdot T) - i$ and segment v with $j - (v \cdot T)$. Equation 10 is a version of LBG lower bound that includes side segments.

$$\mathrm{LBG}(Q^T, E^T, [i, j]) = \sqrt[p]{D(v, v)} \tag{10}$$

$$D(x,y) = l_{x,y} \cdot D_{seg}(q_x^T, e_y^T) + \min \begin{cases} D(x-1,y-1) \\ D(x-1,y) \\ D(x,y-1) \end{cases} \tag{11}$$

$$l_{x,y} = \begin{cases} (u \cdot T) - i & \text{if } x < u \text{ or } y < u \\ j - (v \cdot T) & \text{if } x > v \text{ or } y > v \\ T & \text{otherwise} \end{cases} \tag{12}$$

$$D_{seg}(q_x^T, e_y^T) = \begin{cases} |lq_x^T - ue_y^T|^p & \text{if } lq_x^T > ue_y^T \\ |le_y^T - uq_x^T|^p & \text{if } le_y^T > uq_x^T \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

where $D(u-1,u-1) = 0$ and $D(x,u-1) = D(u-1,y) = \infty$ for $1 \leq x,y \leq t$.

4.2.3 Optimized Leaf Node Sequential Scan. If the lower bound between a query time series and a node's envelope is smaller than the best-so-far distance, we need to sequentially scan a leaf node. Since LBG computes a lower bound for all time series of this node, we first compute a lower bound to the individual time series or more precisely a cascade of lower bounds. Because of its good performance [26], we use a cascade of the two lower bounds LB_KimFL and LB_Keogh (cf. Section 3.5). This might already prune many time series of the node. For the remaining time series, we need to compute the true DTW distance.

4.3 Insert Operation

To insert a new time series, we search the tree for the leaf node with the lowest insertion cost. This section describes: (1) TSEIT's insertion cost functions, (2) the node overflow handling, and (3) TSEIT's parameters.

4.3.1 Insertion Cost. Recall that the LBG results in a lower bound of 0 if the time series completely lies between the upper and lower sequence of the envelope (cf. Section 3.4). To maximize the lower bound, TSEIT minimizes the size of the envelopes, i. e., the area surrounded by the upper and lower sequence. TSEIT's primary insertion cost function is the enlargement of an envelope, i. e., the area it will grow by after inserting the new time series. Equation 14 defines function enlargement(E,C) that returns the size envelope E will grow when inserting time series C.

$$\text{enlargement}^2(E,C) = \sum_{i=1}^{t} \begin{cases} (c_i - le_i)^2 & \text{if } c_i > ue_i \\ (ue_i - c_i)^2 & \text{if } c_i < le_i \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

If there are several envelopes with the same enlargement costs, TSEIT picks the envelope with the least expansion. The expansion of an envelope is the area between its upper and lower sequence. See Equation 15.

$$\text{expansion}(E^T) = \sum_{i=1}^{t} T \cdot \left(eu_i^T - el_i^T \right) \tag{15}$$

While function enlargement considers the growth of an envelope, function expansion considers its total area. In other words, if two envelopes do not need an enlargement, TSEIT chooses the smaller envelope.

4.3.2 Node Overflow Treatment. If an overflow occurs to a node for the first time, TSEIT reinserts time series of this node, to globally minimize envelope expansion. If an overflow occurs the second time, TSEIT splits this node. A node split aims to *locally* minimize envelope expansion. Since its envelope wraps all time series of a node, we initialize the split with the envelope's upper and lower sequence and minimize the sum of the expansion of the resulting envelopes using k-means. We call this heuristic EnvelopeSplit.

4.3.3 TSEIT Parameters. Like any tree, TSEIT has parameters to control the tree development. The first parameter pair sets the minimal and maximal capacity of the leaf nodes. We call these parameters *minNodeSize* and *maxNodeSize*. The second parameter pair defines the minimal and maximal number of child nodes. We call these parameters *minNodeChildren* and *maxNodeChildren*.

4.4 Generalizing Other Approaches

We design TSEIT as a generalization of two state-of-the-art approaches: TWIST [24] and UCR Suite Cascading Lower Bounds (UCR) [26]. TSEIT can simulate the TWIST approach by setting the maximal number of child nodes to infinity. This forces TSEIT to always append new leaf nodes to the root and keep a tree height of two. The leaf nodes simulate the partitions, and the traversal step of the root node simulates the pruning of single partitions. TSEIT can also simulate the UCR approach by setting TSEIT's node size to infinity. This prevents TSEIT from splitting the root and always keeps a single node.

5 EXPERIMENTAL EVALUATION

In this section, we conduct three experiments to gain insights into the benefits and drawbacks of TSEIT. We start with a micro benchmark (Experiment 1) systematically evaluating the influence of parameters, such as the maximum node size, on the response time. We are primarily interested in the response-time robustness of different parameters. In addition, we aim at revealing the most significant parameters influencing the response time and finding a good parameter combination for the remaining experiments. In Experiment 2, we compare the query performance of TSEIT the reference points. The primary objective is to investigate whether TSEIT's combined usage of lower bounding and partitioning, i. e., our generalization, results in significant performance improvements. To this end, we examine how LBG and DTW computations are related to performance differences of the approaches. In the final experiment, we evaluate whether the build times of TSEIT are reasonable. Before we start with the experiments, we describe the experimental setup of all subsequent experiments.

5.1 Experimental Setup

This section describes (1) technical details, (2) used data sets, and (3) data preprocessing.

Technical Details. We run all our benchmarks on a machine having an Intel® Xeon® CPU E5-2630 v3 @ 2.40GHz and 126 GB of RAM. Our machine runs Ubuntu 16.04.4 LTS as operating system. We implement all approaches in Java and execute them on a OpenJDK 64-Bit Server VM in version 1.8.0_181. Our implementation

ensures that all approaches, our TSEIT approach as well as the competitors, keep all their data in memory.

Data Sets. To ensure generality of our findings, we rely on the largest openly available temporal corpus, the Google Books Ngram corpus [22].[2] This corpus is available for 11 languages, reflecting the evolution of the respective language for the last 200 years comprehensively. This corpus is used frequently in the humanities. For our experiments, we use the 2-gram data of the English, German, and Spanish corpus. For each 2-gram, the corpus contains its usage frequency starting at year 1800 and ending with year 2008.

Data Preprocessing. The raw data of the Ngram corpus has several quality issues, such as OCR errors. To this end, we preprocess the data as follows. We remove words from the data set that contain:

Special characters We filter words that contain special characters, e. g., digits. For the definition of special characters, we use the Java function `java.lang.Character.isLetter()`.

Annotations The corpus offers semantic annotations of the words, e. g., whether the word is used as noun or as verb. We filter the annotated words and keep the ones without annotation.

As final preprocessing step, we normalize the time series, to remove the autocorrelation of the time series. Google provides the corpus with the word's absolute usage frequency. With a rising number of published books the number of word usages also rises every year, leading to autocorrelation. We remove this trend by calculating the word's relative usage, i. e., the share of the word usage in the total usage of all words in this year.

5.2 Experiment 1: Parameter Influence

TSEIT is configurable by four parameters: Two specify the node size and two specify a valid number of child nodes. In this subsection, we have the following two objectives: First, we inspect the robustness of the parameters and analyze their effect on the query run time. Second, we aim to find the best parameter settings to benchmark the query performance in Section 5.3.

5.2.1 Influence Analysis. We create data sets of different sizes by randomly subsampling the English corpus. Figure 5 shows the results of our analysis over data sets of different size.

5.2.2 Result Interpretation. Recollect that we have two interests: (1) robustness of kNN query performance, and (2) finding the best parameter settings used subsequently. The second item will also result in first insights whether a combination of lower bounding *and* partitioning minimizes the query time. For instance, in case the best performance is achieved if the maximal node size is close to the data set size, i. e., TSEIT behaves like UCR, it is lower bounding that affects the performance. In contrast, in case every leaf node contains a share of the elements, we see this as an indication that lower bounding *and* partitioning are advantageous.

Parameter Robustness. The results clearly indicate that the most relevant parameter for TSEIT's query performance is *maxNodeSize*. Since we find good values for this parameter independent of the size of the data set (even if the optimal values slightly increase with the size of the data set), we conclude that TSEIT's kNN query performance is robust.

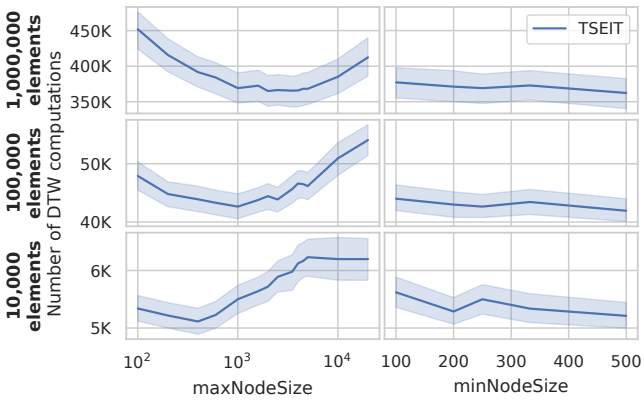

Figure 5: The impact of TSEIT's node size on the number of required DTW computations.

Optimized Parameter Setting. Based on the parameter influence analysis, we found the following parameter setting to be most efficient on the considered data sets, and we therefore rely on it in the subsequent experiments: We set the maximum node size to 1,000 time series and the minimum node size to 250 time series. We set the valid number of node children from 1 to 3, i. e., we allow a maximum of three child nodes per node. This optimized configuration reveals that TSEIT organizes its envelopes hierarchically, without any degeneration yielding behavior like the one of TWIST or UCR. This suggests that our generalization is superior to existing approaches. We investigate this in detail as part of Experiment 2.

So we fix those parameter settings for all further experiments. To have a fair comparison, we set TWIST's partition size to TSEIT's node size. UCR is parameter free.

5.3 Experiment 2: kNN Query Performance

TSEIT's main purpose is minimizing the response time of interval-focused kNN queries. The prior experiment in Section 5.2 gives first indications that our generalized approach is faster than approaches optimizing a single pruning technique. We now examine the query run times in more detail using the following setup.

5.3.1 Benchmark Setup. To provide reproducible, valid results, we provide details of how we have implemented this benchmark. To this end, we first describe the process to select the parameter values of a query. We then specify our performance indicators.

Query Selection. An interval query kNN($k, Q, [i, j]$) depends on the number of neighbors to search k, the query time series Q, and the interval $[i, j]$. This section specifies how we set these parameters.

To have different result sizes, we uniformly vary parameter k from 1 to 10. These sizes yield results interpretable by domain experts, historians in our case. Parameter Q specifies the query time series. Here, we select a random time series from the full corpus. To simulate a real-world workload, we weight the probability of every

[2]The Google Books Ngram corpus is available under http://storage.googleapis.com/books/ngrams/books/datasetsv2.html.

time series with its usage frequency, i. e., more frequent words have a proportionally higher probability to be selected. This reflects the fact that a common word as a query is more likely than, say, a word with a typo in reality. We also vary the start and end point of the query interval $[i, j]$ uniformly, i. e., even if the same time series is selected by chance, the interval is most likely different. Since the run time of DTW depends on the length of the time series and thus the length of the interval query, we have to keep a fixed interval size for our experiments. Otherwise, the interval length dominates the query run time and thus disturbs the variance of our benchmarks. We choose an interval size of 150 years and set DTW's warping window to the same value.

Performance Indicators. Since our main objective is to minimize the query run time, this is our main performance indicator. To explain run time differences and to investigate whether our generalization is the main reason for the observed differences, we rely on two additional implementation-independent measures. The first one quantifies the overhead of the data structure to search for elements, by counting the LBG computations. The rationale is that this computation is the only expensive operation in the traversal (cf. Algorithm 1). The second one counts the distance computations between the query and candidate time series to quantify the effect of partitioning. For both additional indicators, we use the fraction instead of the absolute count to obtain a number unbiased from the size of the data set size that makes different sizes comparable. To sum up, we use the following performance indicators.

Data Structure Search Costs (LBG) We investigate the average fraction of LBG computations necessary to search the data structure, e. g., to traverse the index. A value of 100 % means that the approach needs as many LBG computations as it contains time series. Fewer LBG computations means lower costs to search the data structure and thus should accelerate query processing. Note that, by definition, this measure is 0 for the sequential scan and UCR. This is because these approaches do not use this technique.

Necessary Distance Computations (DTW) We investigate the average fraction of full DTW computations necessary to find the nearest neighbors. A value of 100 % means that the exact distance to each other time series in the data set is computed, i. e., the approach computes as many distances as the sequential scan.

Query Wall-clock Time We measure the time from calling the query routine until it returns. For statistical soundness, we use the mean and variance of executing 100 randomly selected query time series and intervals.

5.3.2 Benchmark Results. Figure 6 plots all performance indicators depending on the number of time series, on three different language corpora. We now describe these results.

LBG Computations. The first row of Figure 6 plots the mean and the confidence band (95 %) of the share of LBG computations necessary to answer a single query. Recall that UCR and the sequential scan do not do any LBG computation. We observe that TSEIT performs very few LBG computations for all sizes of the data set on all three corpora. The value is smaller than 0.4 % in any case. In contrast, TWIST needs 8 to 10 times more LBG computations than TSEIT. TWIST is inefficient for very small data sets, but gets more efficient for larger ones. So we conclude that the working of TSEIT with our optimized parameter settings is significantly different from the one of TWIST. This is another indication that both partitioning and lower bounding are required.

DTW Computations. The second row of Figure 6 plots the mean and the confidence band (95 %) of the number of DTW distance computations necessary to answer a query. Remember that a sequential scan computes for each query point Q the distance to all points in the data set, i. e., has a value of 100 % for this measure. For all other approaches, we observe that they compute only some of the distances. The only exception are very small data sets, where all approaches converge towards a sequential scan. For larger data sets, we clearly see that the most DTW computations are required after applying UCR. Interestingly, the graphs for TSEIT and TWIST have a very similar shape. Observe the small number of distance computations, which usually is between 10 and 20 %. This saves a lot of distance computations since there are up to 5 million time series.

The results regarding this measure allow the following conclusions. First, there is a significant difference between TSEIT and UCR: UCR does not prune groups of irrelevant candidate time series. Second, since TSEIT and TWIST rely on partitioning, both have few DTW distance computations as an effect of LBG pruning. Since the plots of the numbers of DTW computations required for TSEIT and TWIST are very similar, a significant difference in the query run times would mean that indeed lower bounding and partitioning are required.

Query Run Time. The third row of Figure 6 plots the mean and the confidence band (95 %) of the wall-clock time to answer a query. As expected, the run time of the sequential scan grows linearly with the size of the data set. We observe that, for any corpus and size of the data set, all approaches outperform this baseline. On the English corpus, TWIST is faster than UCR, while both have a similar run time on the German and Spanish corpus. Our TSEIT approach shows the best run time on all three corpora. On average, it performs a query 50 % faster than TWIST or UCR.

We conclude that indeed the combination of lower bounding and portioning results in the best response times. These times are not observed for any competitor relying only on one of these techniques. Considering the kNN query times, TWIST is second even if TSEIT clearly outperforms it. To get the full picture, we examine the build times of the index in the next experiment.

5.4 Experiment 3: Index Build Times

The purpose of studying the build time is to evaluate whether a large build time is a counterargument for TSEIT. To this end, we compare TSEIT's build time to the one of the reference points. To avoid biasing our results, we shuffle all time series before inserting them, i. e., we insert them in a random order.

Figure 7 plots the build time of our TSEIT data structure as well as of TWIST, contingent on the size of the data set. UCR and the Sequential scan do not build any index, so they are not part of this experiment. The results suggest that the build times of TWIST are quadratic and might rather be an argument against this approach.

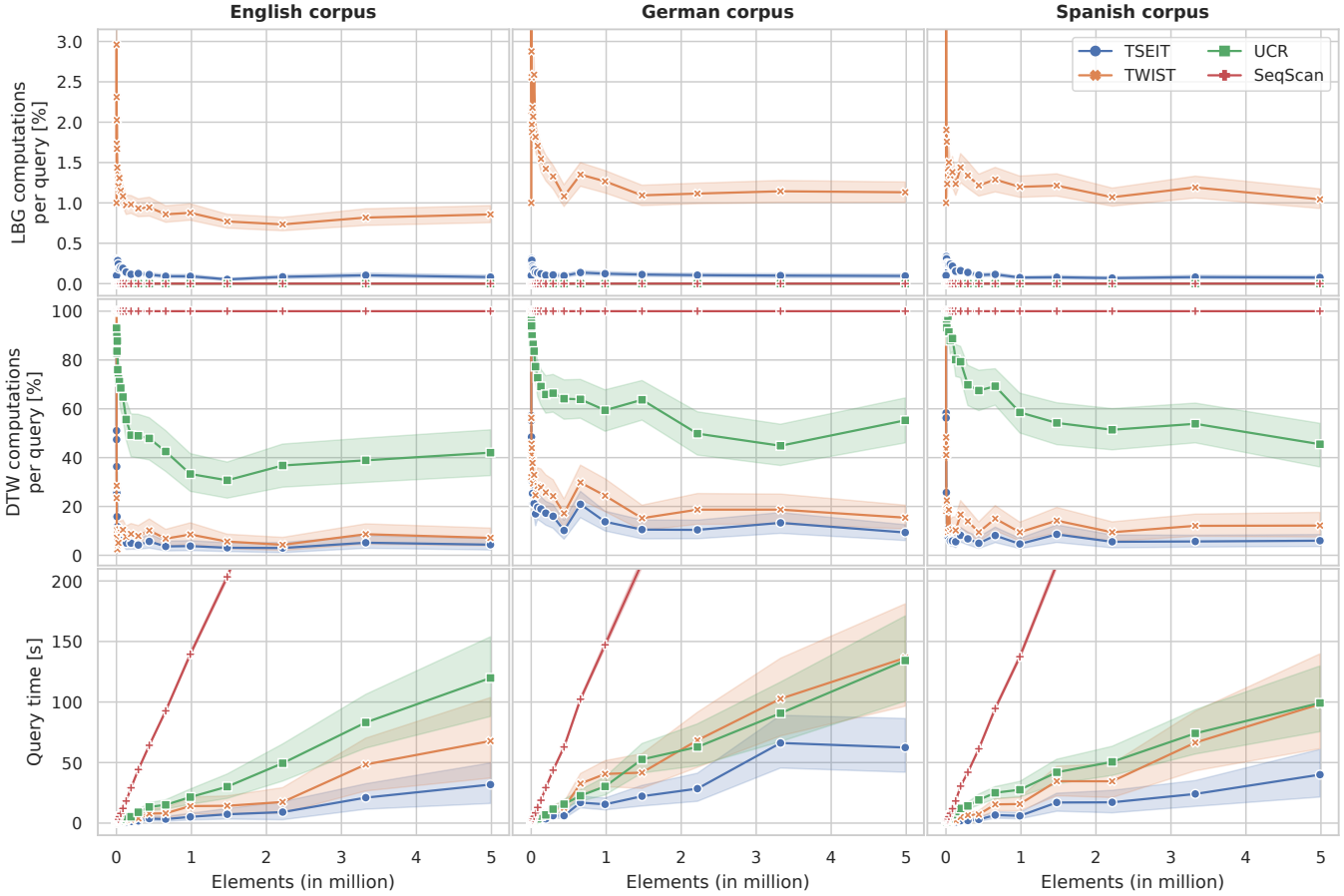

Figure 6: The query performance measurements depending on the number of elements and different corpora.

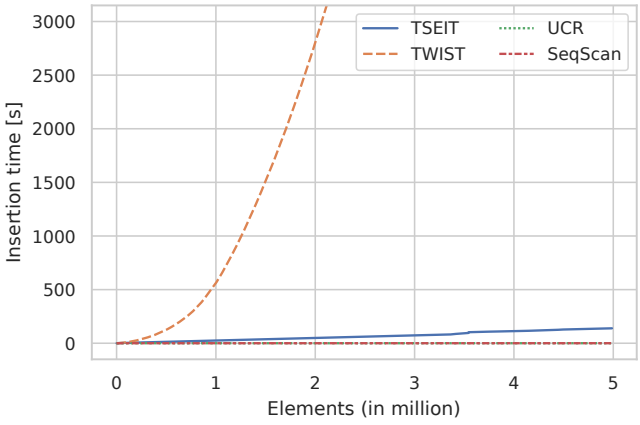

Figure 7: Build times for the English 2-gram corpus.

This is important as TWIST is second behind TSEIT according to Experiment 2.

In contrast to TWIST, the build times for our TSEIT approach appear to be linear. Building the TSEIT index for a data set of 5 million times series in less than 5 minutes is not constraining for most purposes. Thus, our insertion benchmark reveals that TSEIT also is suitable to index large sets of time series.

5.5 Summary

Based on Experiment 2, we conclude that TSEIT consistently results in the lowest run times. This is because TSEIT is able to prune large parts of the data by lower bounding and exclusion of child nodes. On average, TSEIT traverses only small parts of the index, indicated by the small number of LBG computations. Moreover, due to partitioning, it also requires the fewest DTW computations. Our results also indicate that TSEIT's optimal configuration does not converge towards TWIST or UCR.

Finally, studying the build times reveals that indexing even large sets with millions of time series is not a problem for TSEIT. In contrast, the quadratic costs of inserting a time series with the next fastest approach (TWIST) might be problematic for various use cases.

6 CONCLUSIONS

In this work, we address the problem of efficient interval-focused similarity search for time series. We present the idea of hierarchically structured envelopes and, based on it, propose a novel tree-like

data structure named TSEIT. TSEIT is a search tree with an envelope for each node that wraps all time series of the sub-tree of the node. Time series envelopes allow to search the tree given a time series as similarity query and a time interval. Moreover, TSEIT combines various pruning techniques from literature. We are first to systematically consider combinations of pruning techniques for time series. Our evaluation systematically compares the performance with our data structure to the ones of state-of-the-art approaches. The run time of our reference points is either dominated by searching the data structure or by distance computations. According to our experiments, we achieve the best query performance with a compromise between minimizing the costs of searching the data structure and the number of distance computations. Our data structure features a parameter to trade off these two aspects. All this reduces the query times of TSEIT by up to 50 % in comparison to the reference points.

Future Work. Among others, our data structure allows users such as historians or philosophers to study the similarity of word frequencies using the Google Ngram corpus. These users are interested in words whose frequencies change over time due to historical developments and events. The research described here is part of a larger effort enabling philosophers to test their hypotheses empirically.

According to our experience, the DTW distance appears to be sufficiently general regarding the Google Ngram corpus. It may however turn out that users are interested in several distance measures for similarity search, e. g., to compare the results for different measures. We plan to investigate how to evaluate interval-focused kNN queries with several distance measures using a single index. But the specifics of such an index currently are unclear.

REFERENCES

[1] R. Agrawal, C. Faloutsos, and A. Swami. 1993. Efficient similarity search in sequence databases. In *FODO '93*. Springer Berlin Heidelberg, 69–84. https://doi.org/10.1007/3-540-57301-1_5

[2] R. Agrawal, K. Lin, H. Sawhney, and K. Shim. 1995. Fast Similarity Search in the Presence of Noise, Scaling, and Translation in Time-Series Databases. In *VLDB '95*. Morgan Kaufmann Publishers Inc., 490–501.

[3] I. Assent, R. Krieger, F. Afschari, and T. Seidl. 2008. The TS-tree: efficient time series search and retrieval. In *EDBT '08*. ACM Press, 252–263. https://doi.org/10.1145/1353343.1353376

[4] J. Aßfalg, H.-P. Kriegel, P. Kröger, P. Kunath, A. Pryakhin, and M. Renz. 2007. Interval-Focused Similarity Search in Time Series Databases. In *DASFAA '07*. Springer Berlin Heidelberg, 586–597. https://doi.org/10.1007/978-3-540-71703-4_50

[5] O. Brunner, W. Conze, and R. Koselleck (Eds.). 2004. *Geschichtliche Grundbegriffe 1–8*. Klett-Cotta Verlag.

[6] F. Chan, A. Fu, and C. Yu. 2003. Haar wavelets for efficient similarity search of time-series: with and without time warping. *TKDE '03* (2003), 686–705. https://doi.org/10.1109/tkde.2003.1198399

[7] K.-P. Chan and A. Fu. 1999. Efficient time series matching by wavelets. In *ICDE '99*. IEEE, 126–133. https://doi.org/10.1109/icde.1999.754915

[8] B. Chiu, E. Keogh, and S. Lonardi. 2003. Probabilistic discovery of time series motifs. In *KDD '03*. ACM Press, 493–498. https://doi.org/10.1145/956750.956808

[9] Y. Du, C. Jiang, W.-A. Tan, D. Lu, and D. Li. 2008. Effective Subsequence Matching in Compressed Time Series. In *ICPCA '08*. IEEE, 922–926. https://doi.org/10.1109/icpca.2008.4783742

[10] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. 1994. Fast subsequence matching in time-series databases. In *SIGMOD '94*. ACM Press, 419–429. https://doi.org/10.1145/191839.191925

[11] A. Fu, E. Keogh, L. Lau, C. Ratanamahatana, and R. Wong. 2007. Scaling and time warping in time series querying. *The VLDB Journal* (2007), 899–921. https://doi.org/10.1007/s00778-006-0040-z

[12] M.-S. Gil, B.-S. Kim, M.-J. Choi, and Y.-S. Moon. 2015. Fast index construction for distortion-free subsequence matching in time-series databases. In *BIGCOMP '15*. IEEE, 130–135. https://doi.org/10.1109/35021bigcomp.2015.7072822

[13] A. Guttman. 1984. R-trees: A Dynamic Index Structure for Spatial Searching. In *SIGMOD '84*. ACM Press, 47–57. https://doi.org/10.1145/602259.602266

[14] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. 2001. Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases. *Knowledge and Information Systems* (2001), 263–286. https://doi.org/10.1007/pl00011669

[15] E. Keogh and C. Ratanamahatana. 2005. Exact indexing of dynamic time warping. *Knowledge and Information Systems* (2005), 358–386. https://doi.org/10.1007/s10115-004-0154-9

[16] S.-W. Kim, S. Park, and W. Chu. 2001. An index-based approach for similarity search supporting time warping in large sequence databases. In *ICDE '01*. IEEE Computer Society, 607–614. https://doi.org/10.1109/icde.2001.914875

[17] F. Korn, H. Jagadish, and C. Faloutsos. 1997. Efficiently supporting ad hoc queries in large datasets of time sequences. In *SIGMOD '97*. ACM Press, 289–300. https://doi.org/10.1145/253260.253332

[18] M. Krawczak and G. Szkatula. 2010. Time series envelopes for classification. In *IS '10*. IEEE, 156–161. https://doi.org/10.1109/is.2010.5548371

[19] A.-J. Li, Y.-H. Liu, Y.-J. Qi, and S.-W. Luo. 2002. An approach for fast subsequence matching through KMP algorithm in time series databases. In *ICMLC '02*. IEEE, 1292–1295. https://doi.org/10.1109/icmlc.2002.1167412

[20] Q. Li, B. Moon, and I. Lopez. 2004. Skyline index for time series data. *TKDE '04* (2004), 669–684. https://doi.org/10.1109/tkde.2004.14

[21] S.-H. Lim, H. Park, and S.-W. Kim. 2007. Using multiple indexes for efficient subsequence matching in time-series databases. *Information Sciences* (2007), 5691–5706. https://doi.org/10.1016/j.ins.2007.07.004

[22] Y. Lin, J.-B. Michel, E. Aiden, J. Orwant, W. Brockman, and S. Petrov. 2012. Syntactic annotations for the Google Books Ngram Corpus. In *ACL '12*. Association for Computational Linguistics, 169–174.

[23] X.-Y. Liu and C.-L. Ren. 2013. Fast subsequence matching under time warping in time-series databases. In *ICMLC '13*. IEEE, 1584–1590. https://doi.org/10.1109/icmlc.2013.6890855

[24] V. Niennattrakul, P. Ruengronghirunya, and C. Ratanamahatana. 2010. Exact indexing for massive time series databases under time warping distance. *Data Mining and Knowledge Discovery* (2010), 509–541. https://doi.org/10.1007/s10618-010-0165-y

[25] N. Olsen. 2012. *History in the Plural: An Introduction to the Work of Reinhart Koselleck*. Berghahn Books.

[26] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh. 2012. Searching and mining trillions of time series subsequences under dynamic time warping. In *KDD '12*. ACM Press, 262–270. https://doi.org/10.1145/2339530.2339576

[27] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh. 2013. Addressing Big Data Time Series: Mining Trillions of Time Series Subsequences Under Dynamic Time Warping. *TKDD '13* (2013), 1–31. https://doi.org/10.1145/2513092.2500489

[28] J. Ritter, K. Gründer, and G. Gabriel (Eds.). 1971. *Historisches Worterbuch der Philosophie (13 Volume Set) (German Edition)*. Schwabe.

[29] H. Sakoe and S. Chiba. 1978. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing* (1978), 43–49. https://doi.org/10.1109/tassp.1978.1163055

[30] Y. Sakurai, M. Yoshikawa, and C. Faloutsos. 2005. FTW: fast similarity search under the time warping distance. In *PODS '05*. ACM Press, 326–337. https://doi.org/10.1145/1065167.1065210

[31] R. Schneider and H.-P. Kriegel. 1991. The TR*-tree: A new representation of polygonal objects supporting spatial queries and operations. In *CG '91*. Springer Berlin Heidelberg, 249–263.

[32] M. Schäler, A. Grebhahn, R. Schröter, S. Schulze, V. Köppen, and G. Saake. 2013. QuEval: beyond high-dimensional indexing à la carte. *VLDB Endowment* (2013), 1654–1665. https://doi.org/10.14778/2556549.2556551

[33] E. Vidal, F. Casacuberta, J. Benedi, M. Lloret, and H. Rulot. 1988. On the verification of triangle inequality by dynamic time-warping dissimilarity measures. *Speech Communication* (1988), 67–79. https://doi.org/10.1016/0167-6393(88)90022-2

[34] R. Weber, H.-J. Schek, and S. Blott. 1998. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *VLDB '98*. Morgan Kaufmann Publishers Inc., 194–205.