Fast Retrieval of Similar Subsequences in Long Sequence Databases

Sanghyun Park, Dongwon Lee, Wesley W. Chu
Department of Computer Science
University of California, Los Angles, California 90095 USA
{shpark,dongwon,wwc}@cs.ucla.edu

Abstract

Although the Euclidean distance has been the most popular similarity measure in sequence databases, recent techniques prefer to use high-cost distance functions such as the time warping distance and the editing distance for wider applicability. However, if these distance functions are applied to the retrieval of similar subsequences, the number of subsequences to be inspected during the search is quadratic to the average length $\bar{L}$ of data sequences. In this paper, we propose a novel subsequence matching scheme, called the aligned subsequence matching, where the number of subsequences to be compared with a query sequence is reduced to linear to $\bar{L}$. We also present an indexing technique to speed-up the aligned subsequence matching using the similarity measure of the modified time warping distance. The experiments on the synthetic data sequences demonstrate the effectiveness of our proposed approach; ours consistently outperformed the sequential scanning and achieved up to 6.5 times speed-up.

1. Introduction

Similarity search in sequence databases plays an important role in many application domains such as information retrieval and data mining. Recently, many techniques [1, 6, 7] have been proposed to support the fast retrieval of similar sequences using the Euclidean distance metric. However the Euclidean distance metric has the following problems: 1) it cannot be applied to sequences of different lengths or different sampling rates, and 2) it is very sensitive to small distortion on the time axis. To remedy these problems, [4] used the modified editing distance and [14] used the time warping distance, concentrating on the whole sequence matching.

However, if we use the editing distance or the time warping distance for subsequence matching, the number of subsequences to be inspected during the search becomes quadratic to the average length $\bar{L}$ of data sequences. Thus, when $\bar{L}$ is very large, we may suffer from severe performance degradation.

In this paper, we propose a novel sequence matching scheme, called the *aligned subsequence matching*, where the number of subsequences to be compared with a query sequence is reduced to $O(\bar{L})$. In aligned subsequence matching, sequences are divided by piece-wise segments and only those subsequences starting and ending at segment boundaries are inspected during the search. We also present an indexing technique to support the fast retrieval of similar aligned subsequences using the similarity measure of the modified time warping distance. Our indexing technique is summarized as follows: First, we extract a feature vector from each segment and group similar feature vectors together. Then, we convert each segment into a symbol of the corresponding group. Finally, from sequences of symbols, we construct the generalized suffix tree (GST). At search time, the GST is traversed to find candidate subsequences which are *possibly* with the distance tolerance from the query sequence. Aligned subsequences that are actually within the distance tolerance are obtained after post-processing.

The rest of this paper is organized as follows. In Section 2, we provide a brief overview of the related work on sequence matching problems. Section 3 contains the definition and the similarity measure of aligned subsequence matching. The indexing construction method and the query processing algorithm are presented in Sections 4 and 5, respectively. Finally, our proposal is verified by the experimental results in Section 6 followed by concluding remarks in Section 7.

2. Related work

Several approaches for fast retrieval of similar sequences have been recently proposed. In [1], sequences are converted into the frequency domain by the Discrete Fourier Transform and are subsequently mapped into multi-dimensional points that are managed by an R^*-tree. This technique can be extended to locate similar subsequences [6]. Since both approaches are based on the Euclidean distance metric, sequences of different lengths or different sampling rates cannot be matched.

The access methods of [4, 14, 8] permit the matching of sequences of different lengths. [4] presents the modified version of edit distance, considering two sequences matching if a majority of elements match. In [14], the time warping distance is used as a similarity measure with the two-step filtering process: a FastMap index filter followed by a lower-bound distance filter. Since the modified editing distance and the time warping distance are expensive, both [4] and [8] focus on whole sequence matching. [8] presents an access method for subsequence matching with the similarity measure of the time warping distance. Using a generalized suffix tree as an index structure and two lower-bound distance functions as index filters, [8] retrieves similar subsequences without false dismissals. However, the computation complexity of [8] is quadratic to the average length of data sequences.

3. Aligned subsequence matching

We assume that sequences consist of a series of real numbers. We denote a sequence $\langle x_1, ..., x_L \rangle$ as $\vec{x}$. Table 1 shows a list of notations used in the paper. To reduce the number of subsequences to be compared with a query sequence, in this section, we suggest the aligned subsequence matching. In the aligned subsequence matching, sequences are divided into piece-wise segments and only those subsequences $\vec{x}$[p:r] satisfying the following conditions are inspected during the search:

1. p is the starting position of a segment.
2. r is the ending position of the same segment or its following segments.
3. The number of segments in $\vec{x}[p:r]$ is the same as that of segments in a query sequence.

We call a subsequence satisfying the first two conditions an *aligned subsequence*. Let us use the notation C for the average number of elements in a segment. Then there are $\frac{|\vec{x}|}{C}$ segments in a data sequence $\vec{x}$ and $\frac{|\vec{q}|}{C}$ segments in a query sequence $\vec{q}$. Therefore, in a data sequence $\vec{x}$, the number of aligned subsequences that have $\frac{|\vec{q}|}{C}$ segments is $\frac{|\vec{x}|}{C} - \frac{|\vec{q}|}{C} + 1$. Thus the number of subsequences satisfying the above three conditions is linear to the length of a data sequence.

Now, we describe the method to get the segmented representation of a sequence and the distance function to measure similarities of aligned subsequences.

3.1. Segmentation

To obtain a series of piece-wise segments from a sequence, we take the following segmentation approach: First, the algorithm scans a sequence and records all the peak points. $\vec{x}[p]$ is a peak point if $\vec{x}[p-1] \leq \vec{x}[p] \geq \vec{x}[p+1]$ or $\vec{x}[p-1] \geq \vec{x}[p] \leq \vec{x}[p+1]$. Then, the algorithm finds the peak point that has the largest deviation from the interpolation line connecting the two end points of a sequence. If the found peak point satisfies the semantic constraints such as the minimum interval constraint and the minimum value-change constraint, the sequence is divided into two segments at that peak point. The algorithm is executed recursively until some threshold is exceeded. For details, refer to [9]. Our segmentation algorithm is an improved version of the one proposed in [11] in two aspects:

1. Let L be the number of elements in a sequence and P be the number of peak-point elements in the same sequence. Our approach has the algorithmic complexity $O(L + P^2)$ while the approach in [11] is $O(LP)$. When L is very large and $P \ll L$ (which is very likely the case in practice), ours is more efficient than the algorithm in [11].
2. Ours considers semantic constraints of a sequence. For instance, in stock trading data sequences, if only segments whose change rates are more than 5% are of interest, then our algorithm can filter out those segments whose increase or decrease rates are less than 5% at early stage. Semantic constraints can also be used to eliminate noises.

We denote the sequence of segments obtained from $\vec{x}$ as $\vec{x}^S$. The length of $\vec{x}^S$ is much smaller than that of $\vec{x}$. The *compaction ratio (C)* can be expressed as $C = |\vec{x}|/|\vec{x}^S|$. C is also considered as the average number of elements in a segment.

Example 1: Let us consider a data sequence $\vec{x} = \langle 4, 5, 8, 8, 8, 8, 9, 11, 8, 4, 3, 7, 10 \rangle$. $\vec{x}$ is segmented to $\vec{x}^S = \langle \langle 4, 5, 8, 8, 8, 8, 9, 11 \rangle, \langle 8, 4, 3 \rangle, \langle 7, 10 \rangle \rangle$. Then $\vec{x}^S[1] = \langle 4, 5, 8, 8, 8, 8, 9, 11 \rangle$, $\vec{x}^S[2] = \langle 8, 4, 3 \rangle$, and $\vec{x}^S[3] = \langle 7, 10 \rangle$. Since $|\vec{x}| = 13$ and $|\vec{x}^S| = 3$, $C = 13/3 = 4.3$. ■

3.2. Similarity measure

Similarity measures for aligned subsequences are based on their segmented representations. Since the lengths of segments to be compared may be different, we propose to use the modified time warping distance as a similarity measure. Given two aligned subsequences $\vec{x}$ and $\vec{y}$, the distance function is defined as follows:

$$D_{sim}(\vec{x}, \vec{y}) = \sum_{i=1}^{N} D_{tw}(\vec{x}^S[i], \vec{y}^S[i])$$

Notation	Description
M	number of data sequences in a database.
ε	distance tolerance given by user.
$\langle\rangle$	empty sequence.
$\vec{x}, \vec{y}$	sequences of real numbers.
$\|\vec{x}\|$	number of elements in $\vec{x}$.
$\vec{x}[p]$	p-th element of $\vec{x}$.
$\vec{x}[p:r]$	subsequence of $\vec{x}$ containing elements from p to r.
$\vec{x}[p:-]$	suffix of $\vec{x}$ starting from p.
$\vec{x}^S$	sequence of segments derived from $\vec{x}$.
$\vec{x}^F$	sequence of feature vectors derived from $\vec{x}$.
$\vec{x}^C$	sequence of symbols derived from $\vec{x}$.
$\vec{\mu}, \vec{\nu}$	sequences of ranges.
$\vec{\alpha}, \vec{\beta}$	segments extracted from sequences of real numbers
A, B	symbols representing a set of segments

Table 1. List of notations.

where N is the number of segments contained in $\vec{x}$ and $\vec{y}$, and $\mathcal{D}_{tw}(\vec{x}^S[i], \vec{y}^S[i])$ is the time warping distance between two segments. This formula can be rephrased as "the distance between two aligned subsequences is the sum of the time warping distances between each pair of segments". The time warping distance between two segments, $\vec{\alpha}$ and $\vec{\beta}$, is defined as follows [10]:

$$
\begin{aligned}
D_{tw}(\langle\rangle, \langle\rangle) &= 0 \\
D_{tw}(\vec{\alpha}, \langle\rangle) &= D_{tw}(\langle\rangle, \vec{\beta}) = \infty \\
D_{tw}(\vec{\alpha}, \vec{\beta}) &= |\vec{\alpha}[0] - \vec{\beta}[0]| \quad + \\
& \quad min \begin{cases} D_{tw}(\vec{\alpha}, \vec{\beta}[2:-]) \\ D_{tw}(\vec{\alpha}[2:-], \vec{\beta}) \\ D_{tw}(\vec{\alpha}[2:-], \vec{\beta}[2:-]) \end{cases}
\end{aligned}
$$

$D_{tw}(\vec{\alpha}, \vec{\beta})$ can be efficiently calculated with computation complexity $O(|\vec{\alpha}||\vec{\beta}|)$ using a dynamic programming technique based on the recurrence relation [2].

Example 2: Suppose we want to find all the aligned subsequences of a data sequence $\vec{x} = \langle 4, 5, 8, 8, 8, 8, 9, 11, 8, 4, 3, 7, 10\rangle$ that are similar to a query sequence $\vec{q} = \langle 3, 1, 0, 1, 3\rangle$ within a distance tolerance $\varepsilon = 25$. Further suppose that $\vec{x}$ and $\vec{q}$ are segmented to $\vec{x}^S = \langle\langle 4, 5, 8, 8, 8, 8, 9, 11\rangle, \langle 8, 4, 3\rangle, \langle 7, 10\rangle\rangle$ and $\vec{q}^S = \langle\langle 3, 1, 0\rangle, \langle 1, 3\rangle\rangle$, respectively. Since $\vec{x}^S$ and $\vec{q}^S$ have 3 and 2 segments, respectively, sequence $\vec{q}^S = \langle\vec{q}^S[1], \vec{q}^S[2]\rangle$ can possibly be compared with only two combinations of segments of $\vec{x}^S$; $\langle\vec{x}^S[1], \vec{x}^S[2]\rangle$ and $\langle\vec{x}^S[2], \vec{x}^S[3]\rangle$. The distances can be computed as follows:

- $D_{sim}(\langle 4, 5, 8, 8, 8, 8, 9, 11, 8, 4, 3\rangle, \langle 3, 1, 0, 1, 3\rangle)$
  $= D_{tw}(\vec{x}^S[1], \vec{q}^S[1]) + D_{tw}(\vec{x}^S[2], \vec{q}^S[2])$
  $= 44 + 8 = 52$.
- $\mathcal{D}_{sim}(\langle 8, 4, 3, 7, 10\rangle, \langle 3, 1, 0, 1, 3\rangle)$
  $= \mathcal{D}_{tw}(\vec{x}^S[2], \vec{q}^S[1]) + \mathcal{D}_{tw}(\vec{x}^S[3], \vec{q}^S[2])$
  $= 11 + 13 = 24$.

Since only $\mathcal{D}_{sim}(\langle 4, 3, 2, 1, 0, 1, 3\rangle, \langle 3, 1, 0, 1, 3\rangle)$ has a distance within the given ε, the aligned subsequence that matches a query sequence $\vec{q}$ is the $\langle 8, 4, 3, 7, 10\rangle$. ■

4. Index construction

Although the aligned subsequence matching is significantly faster than the conventional subsequence matching (i.e., linear vs. quadratic), we can further improve the search time by adopting a sophisticated indexing method. Figure 1 shows the three steps of our index construction method.

4.1. Feature extraction

We extract a set of representative features, called a feature vector, from each segment. For a segment $\vec{\alpha}$, the 5-tuple feature vector is represented as $(L, V_1, V_L, \delta_+, \delta_-)$, where $L = |\vec{\alpha}|$, $V_1 = \vec{\alpha}[1]$, $V_L = \vec{\alpha}[L]$, and δ_+ and δ_- are the positive and the negative maximum deviations from the interpolation line connecting the two end points $(1, V_1)$ and (L, V_L), respectively. For instance, consider three segments $\vec{\alpha}_1 = \langle 4,5,8,8,8,8,9,11\rangle$, $\vec{\alpha}_2 = \langle 8,4,3\rangle$, $\vec{\alpha}_3 = \langle 7,10\rangle$. Their corresponding feature vectors are shown in Table 2.

4.2. Categorization

We generate a set of categories from feature vectors obtained in the previous step. Using the categorization method of multiple-attribute type abstraction hierarchy (MTAH [5]), we classify the similar feature vectors into the

Figure 1. Three steps for index construction.

Segment	Feature Vector					Interpolation line
	L	V_1	V_L	δ_+	δ_-	
$\vec{\alpha}_1$ = ⟨4,5,8,8,8,8,9,11⟩	8	4	11	max{0,0,2,1,0,0,0,0} = 2	max{0,0,0,0,0,1,1,0} = 1	y = x + 3
$\vec{\alpha}_2$ = ⟨8,4,3⟩	3	8	3	max{0,0,0} = 0	max{0,1.5,0} = 1.5	y = –2.5x + 10.5
$\vec{\alpha}_3$ = ⟨7,10⟩	2	7	10	max{0,0} = 0	max{0,0} = 0	y = 3x + 4

Table 2. Feature vectors from three segments.

same category and assign a unique symbol to each category. MTAH is a data-driven multiple-level categorization hierarchy that uses *relaxation error* as a goodness measure of categories. MTAH has the following benefits: 1) The algorithm considers both value and frequency distributions, thus generating more accurate categories than equal-length interval categorization methods; and 2) MTAH is easier to implement than maximum-entropy categorization methods.

Categories are represented as C = ($[L.lb, L.ub]$, $[V_1.lb, V_1.ub]$, $[V_L.lb, V_L.ub]$, $[\delta_+.lb, \delta_+.ub]$, $[\delta_-.lb, \delta_-.ub]$). Table 3 is an example that shows three category symbols and their feature ranges.

After obtaining a set of categories, we convert every segment to a symbol of the corresponding category. Thus, if the feature vector from a segment $\vec{\alpha}$ is included in the symbol A, $\vec{\alpha}$ is replaced with A. For instance, $\vec{x}^S$ = $\langle\langle 4, 5, 8, 8, 8, 8, 9, 11\rangle\rangle$, $\langle 8, 4, 3\rangle$, $\langle 7, 10\rangle\rangle$ can be converted to $\vec{x}^C$ = $\langle A, B, C\rangle$ according to Table 3.

4.3. Suffix tree construction

Once we have converted sequences of real numbers to sequences of symbols, we propose to use a generalized suffix tree (GST [3, 13]) as an index structure for fast aligned subsequence matching. A GST has the benefits such that 1) It is a good structure especially for subsequence matching since all possible suffixes of the given sequence is maintained in a GST and 2) It does not assume any geometry or any distance function. Thus, it guarantees the absence of false dismissals even with the time warping distance if lower-bound distances are used to filter out dissimilar subsequences in index space.

Let us present the definition and the internal structure of a GST. A *suffix tree* [12] is a compact representation of a trie corresponding to the suffixes of a given string where all nodes with one child are merged with their parents. A GST is an extension of the suffix tree allowing for multiple sequences to be stored in the same tree. $\vec{x}[p : -]$ is expressed by a leaf node labeled with $(id(\vec{x}), p)$, where $id(\vec{x})$ is a unique identifier for the given sequence $\vec{x}$ and p is the offset from which the suffix starts. The edges are labeled with subsequences such that the concatenation of the edge labels on the path from the root to the leaf $(id(\vec{x}), p)$ becomes $\vec{x}[p : -]$. We use the notation $label(N_i, N_j)$ for the concatenated labels on the path from N_i to N_j.

To build a suffix tree from multiple sequences, we use an incremental disk-based GST construction method proposed in [3]. Two GSTs, representing two disjoint sets of sequences, are merged to produce a single GST by pre-order traversal of both GSTs and combining the paths corresponding to common subsequences. The construction of GST for M data sequences whose average length is $\bar{L}$ has algorithmic complexity $O(M\bar{L})$ [12].

5. Query processing

Given a set of data sequences, the query sequence $\vec{q}$ and the distance tolerance ε, we want to find all aligned subsequences $\vec{x}$ satisfying the following conditions:

1. The number of elements in $\vec{x}^S$ is the same as that of elements in $\vec{q}^S$ (i.e., $|\vec{x}^S| = |\vec{q}^S|$).
2. The modified time warping distance between $\vec{x}$ and $\vec{q}$ is within ε (i.e., $D_{sim}(\vec{x}, \vec{q}) \leq \varepsilon$).

In this section, we describe our query processing algorithm SearchSubSequence that consists of three steps (Figure 2): pre-processing, index searching and post-processing.

5.1. Pre-processing

The query sequence $\vec{q}$ is converted to the sequence of segments $\vec{q}^S$. Then, a 5-tuple feature vector described in Section 4 is extracted from each segment. Finally, each feature vector is replaced with a symbol of the corresponding

Symbol	$L.lb$	$L.ub$	$V_1.lb$	$V_1.ub$	$V_L.lb$	$V_L.ub$	$\delta_+.lb$	$\delta_+.ub$	$\delta_-.lb$	$\delta_-.ub$
A	6	8	3	5	10	13	1.5	2.5	0	2
B	2	3	7	9	2	4	0	1.5	0.5	3
C	2	3	7	9	4	10	0	1.5	0	1.5

Table 3. Three category symbols and their feature ranges.

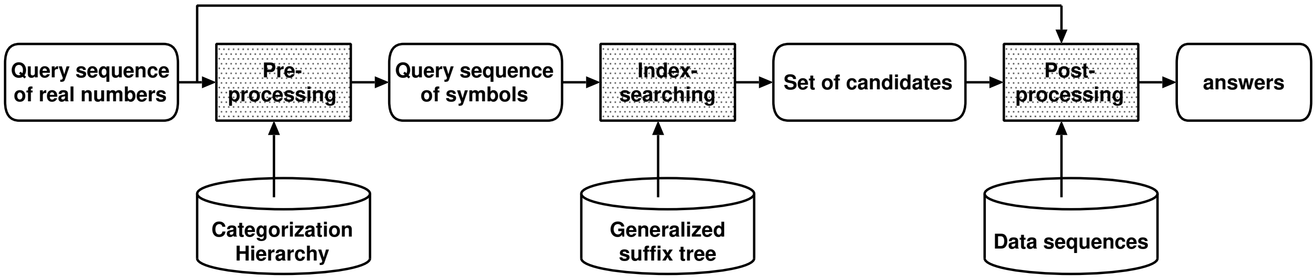

Figure 2. Three steps of query processing algorithm SearchSubSequence

category using the categorization hierarchy built at the stage of index construction. The output of this step is the query sequence of symbols $\vec{q}^C$.

5.2. Index-searching

Depth-first traversal is performed on a GST to find a set of candidates whose distances to the query sequence are *possibly* within the distance tolerance ε.

Our index-searching algorithm IndexSearch is shown in Algorithm 1. For simpler description, we assume that each edge is labeled with a single symbol. The algorithm starts with three inputs: root node N_0, the query sequence of symbols $\vec{q}^C$, and the distance tolerance ε.

When the algorithm visits a node, it computes the distance between each child label and the first element of $\vec{q}^C$ using the distance function $D_{tw-lb}()$ that is designed to satisfy the lower-boundness [1]. The detail description of $D_{tw-lb}()$ is given later in this subsection. Let the distance between the child label and the first element of $\vec{q}^C$ be $dist$. If $dist$ is larger than ε, the algorithm discards the child node and inspects the next child node. Otherwise, either one of the following operations is executed according to the length of $\vec{q}^C$.

1. When $|\vec{q}^C| = 1$, the concatenated labels on the path from the root node to the child node is inserted into a set of candidates.

2. When $|\vec{q}^C| > 1$, the algorithm IndexSearch is called recursively with the modified three inputs: the child node, the query subsequence of symbols $\vec{q}^C[2:-]$, the adjusted distance tolerance $\varepsilon - dist$.

```
Input   : node N, query q^C, distance-tolerance ε
Output  : candSet

candSet ← φ;
CN ← GetChildren(N);
for i ← 1 to | CN | do
    dist ← D_{tw-lb}(label(N, CN_i), q^C[1]);
    if dist ≤ ε then
        if |q^C| = 1 then
            insert label(rootNode, CN_i) into candSet;
        else
            candSet ← candSet ∪ IndexSearch(CN_i, q^C[2 : −], ε − dist);
return candSet;
```

Algorithm 1: IndexSearch

The function $D_{tw-lb}()$ defines the distance of two symbols. Before presenting the definition of $D_{tw-lb}()$, we first describe the method to derive the information on segments included in the specific symbol.

5.2.1. Converting symbols to sequences of ranges. Given a segment (and thus its feature vector), the corresponding symbol can be easily found by looking up the mapping table that stores the lower and the upper bound values of each feature. However, going the other way around, given a symbol, it is difficult to find out all segments included in the symbol without scanning all sequences contained in a database. Nevertheless, using the bound-values of each feature, we still extract the following information on all segments included in the symbol.

- possible length range
- possible value range for each element

That is, given a symbol A, we can derive the sequence of ranges $\vec{\mu}_A = \langle (lb_1, ub_1), ..., (lb_H, ub_H) \rangle$ where H is the maximum length ($A.L.ub$) of segments included in A. Then $\vec{\mu}_A[1:r]$ represents all segments of A that have the length of r ($A.L.lb \le r \le A.L.ub$). Let $minIP(i)$ and $maxIP(i)$ be the minimum and the maximum interpolation values of i-th element, respectively. Then, possible value range of each element, lb_i and ub_i, is computed as follows: (A is omitted for briefness.)

$$
\begin{aligned}
lb_i &= \begin{cases} V_1.lb & (i = 1) \\ minIP(i) - \delta_{-}.ub & (1 < i < H) \\ V_L.lb & (i = H) \end{cases} \\
ub_i &= \begin{cases} V_1.ub & (i = 1) \\ maxIP(i) + \delta_{+}.ub & (1 < i < H) \\ V_L.ub & (i = H) \end{cases}
\end{aligned}
$$

Example 3: Let us compute the sequence of ranges $\vec{\mu}_B$ for the symbol B in Table 3. The maximum length of $\vec{\mu}_B$ is 3 since $L.ub$ = 3. Therefore, $\vec{\mu}_B = \langle (lb_1, ub_1), (lb_2, ub_2), (lb_3, ub_3) \rangle$. By using the above formula, $(lb_1, ub_1) = (7, 9)$ and $(lb_3, ub_3) = (2, 4)$ can be easily computed. The computation of (lb_2, ub_2) is more complicated. $lb_2 = minIP(2) - \delta_{-}.ub = 4.5 - 3 = 1.5$. $ub_2 = maxIP(2) + \delta_{+}.ub = 6.5 + 1.5 = 8$. Thus, $\vec{\mu}_B = \langle (7, 9), (1.5, 8), (2, 4) \rangle$. This can be interpreted as "The segments included in the symbol B can have starting value between 7 and 9, and ending value between 2 and 4. Also they may have intermediate value between 1.5 and 8. ■

5.2.2. Modified similarity measures. Since the modified time warping distance function defined in Section 3 is based on sequences of segments, it is not directly applicable to the index-searching algorithm IndexSearch which operates on sequences of symbols. To remedy this problem, we introduce three distance functions.

- $D_{sr-lb}(\vec{\mu}, \vec{\nu})$ computes the lower-bound distance between two sequences of ranges, $\vec{\mu}$ and $\vec{\nu}$.
- $D_{tw-lb}(A, B)$ computes the lower-bound distance between two symbols, A and B.
- $D_{sim-lb}(\vec{x}, \vec{y})$ computes the lower-bound distance between two aligned subsequences, $\vec{x}$ and $\vec{y}$.

Let us first define the lower-bound distance function for sequences of ranges. Given two sequences of ranges $\vec{\mu}$ and $\vec{\nu}$, the lower-bound distance function $D_{sr-lb}(\vec{\mu}, \vec{\nu})$ is defined as follows:

$$
\begin{aligned}
\mathcal{D}_{sr-lb}(\langle\rangle, \langle\rangle) &= 0 \\
\mathcal{D}_{sr-lb}(\vec{\mu}, \langle\rangle) &= \mathcal{D}_{sr-lb}(\langle\rangle, \vec{\nu}) = \infty \\
\mathcal{D}_{sr-lb}(\vec{\mu}, \vec{\nu}) &= \mathcal{D}_{sr-lb}(\vec{\mu}[1], \vec{\nu}[1]) \;+ \\
&\quad min \begin{cases} \mathcal{D}_{sr-lb}(\vec{\mu}, \vec{\nu}[2:-]) \\ \mathcal{D}_{sr-lb}(\vec{\mu}[2:-], \vec{\nu}) \\ \mathcal{D}_{sr-lb}(\vec{\mu}[2:-], \vec{\nu}[2:-]) \end{cases}
\end{aligned}
$$

$$
\mathcal{D}_{rg}(\mu, \nu) = \begin{cases} 0 & (\mu \text{ and } \nu \text{ overlap}) \\ \nu.lb - \mu.ub & (\mu.ub < \nu.lb) \\ \mu.lb - \nu.ub & (\mu.lb > \nu.ub) \end{cases}
$$

where μ and ν are value ranges corresponding to $\vec{\mu}[1]$ and $\vec{\nu}[1]$, respectively. A value range is denoted as $[lb, ub]$. $D_{sr-lb}(\vec{\mu}, \vec{\nu})$ returns the possible smallest distance between two segments; one segment included in $\vec{\mu}$ and the other segment included in $\vec{\nu}$.

Now, let us define the lower-bound distance function for two symbols. Let $\vec{\mu}_A$ and $\vec{\mu}_B$ be the sequences of ranges derived from two symbols A and B, respectively. Then, given two symbols A and B, the lower-bound distance function $D_{tw-lb}(A, B)$ is defined as follows:

$$
\mathcal{D}_{tw-lb}(A, B) = min\{\forall_i \forall_j (\mathcal{D}_{sr-lb}(\vec{\mu}_A[1:i], \vec{\mu}_B[1:j]))\}
$$

where $A.L.lb \le i \le A.L.ub$ and $B.L.lb \le j \le B.L.ub$. Thus, the distance between symbols A and B can be considered as the possible smallest distance between two segments; one segment in any sequences of ranges derived from A and the other segment in any sequences of ranges derived from B.

Having defined the distance function for symbols, we now define the lower-bound distance function for two aligned subsequences. Give two aligned subsequences, $\vec{x}$ and $\vec{y}$, the lower-bound distance function $D_{sim-lb}(\vec{x}, \vec{y})$ is defined as follows:

$$
D_{sim-lb}(\vec{x}, \vec{y}) = \sum_{i=1}^{N} D_{tw-lb}(\vec{x}^C[i], \vec{y}^C[i])
$$

where N is the number of elements in $\vec{x}^C$ and $\vec{y}^C$. This formula can be rephrased as "the lower-bound distance between two aligned subsequences is the sum of the lower-bound distance between each pair of symbols".

Theorems 1 and 2 state the lower-boundness of $D_{tw-lb}()$ and $D_{sim-lb}()$, respectively. Based on these two theorems, we can guarantee that the index-searching algorithm IndexSearch does not generate false dismissals.

Theorem 1: (Lower-Boundness of $D_{tw-lb}()$)
For any two segments $\vec{\alpha}$ and $\vec{\beta}$, and their corresponding symbols A and B, the following inequalities holds:

$$\mathcal{D}_{tw-lb}(A, B) \quad \leq \quad D_{tw}(\vec{\alpha}, \vec{\beta})$$

Proof: The proof is given in [9]. ■

Theorem 2: (Lower-Boundness of $D_{sim-lb}()$)
For any two aligned subsequences, $\vec{x}$ and $\vec{y}$, the following inequalities holds:

$$\mathcal{D}_{sim-lb}(\vec{x}, \vec{y}) \quad \leq \quad D_{sim}(\vec{x}, \vec{y})$$

Proof: The proof is given in [9]. ■

5.3. Post-processing

For each candidate answer obtained from the previous step, the actual aligned subsequences are retrieved from a database and their distances from the query sequence are computed using the modified time warping distance function $D_{sim}()$. Then aligned subsequences that are actually within the distance tolerance are reported as final answers.

5.4. Analysis of the algorithms

Let $\bar{L}$ be the average length of the data sequences, and C be the average number of elements in segments (i.e., compaction ratio). Let us first examine the complexity of sequential scanning.

The computation complexity for calculating the time warping distance of two segments is $O(C^2)$. The complexity for measuring the modified time warping distance between a query sequence and an aligned subsequence is $O(\frac{C^2|\vec{q}|}{C}) = O(C|\vec{q}|)$. Since the average number of the aligned subsequences with $\frac{|\vec{q}|}{C}$ segments in a data sequence is $(\frac{\bar{L}}{C} - \frac{|\vec{q}|}{C} + 1)$, the complexity for processing M sequences is $O(M|\vec{q}|(\bar{L} - |\vec{q}| + C))$. If $\bar{L} \gg |\vec{q}|$, the complexity becomes $O(M|\vec{q}|\bar{L})$.

Now, let us consider the complexity of the proposed query processing algorithm SearchSubSequence. The complexity for computing $\mathcal{D}_{tw-lb}()$ is the same as $\mathcal{D}_{tw}()$, but is reduced to $O(1)$ if we pre-compute the distances of all pair of symbols and keep them in table. Then the complexity of SearchSubSequence is $O(\frac{M|\vec{q}|(\bar{L}-|\vec{q}|+C)}{C^2 R} + NC|\vec{q}|)$ where R (≥ 1) is the reduction factor saved by sharing edges of the GST and N is the number of the aligned subsequences requiring post-processing. If $\bar{L} \gg |\vec{q}|$, the complexity becomes $O(\frac{M\bar{L}|\vec{q}|}{C^2 R} + NC|\vec{q}|)$.

6. Experimental Results

We used a dataset from UC Irvine KDD Archive (http://kdd.ics.uci.edu) for testing. The dataset, called "Pseudo Periodic Synthetic Time Series", is specifically designed for testing indexing schemes in time series databases. The actual sequence is generated by following function:

$$\vec{x} = \sum_{i=3}^{7} \frac{1}{2^i} sin(2\pi(2^{2+i} + rand(2^i))\vec{t})$$

where $0 \leq \vec{t} \leq 1$. We generated 90 sequences (each is 10,000-datapoints) out of the original 9 sequences and randomly extracted query sequences from the 10-th sequence. Figure 3 shows the experimental results. Our scheme consistently outperformed the sequential scanning and achieved up to 6.5-time speed-up (653%).

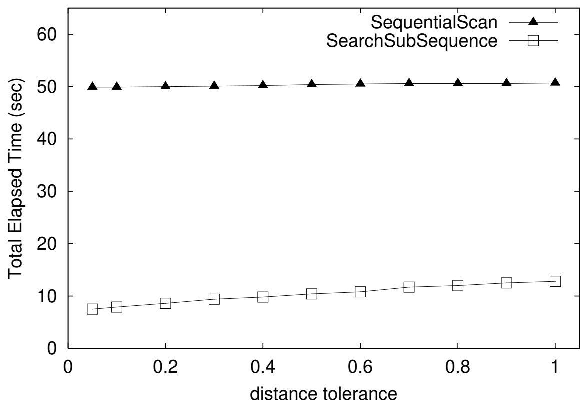

Figure 3. Performance comparison between SequentialScan **and** SearchSubSequence **algorithms.**

7. Conclusion

In this paper, we proposed the aligned subsequence matching whose running time is linear to the total number of and the average length of data sequences. To speed up the aligned subsequence matching, we also presented an efficient indexing method that is based on the generalized suffix tree (GST) and lower-bound distance functions.

The contributions of our works are: 1) the aligned subsequence matching and its similarity measure, 2) the efficient and systematic segmentation algorithm, 3) the compact index construction method using feature extraction and categorization, and 4) the effective query processing algorithm based on lower-bound distance functions.

References

[1] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In *Proc. FODO*, 1993.

[2] D. J. Berndt and J. Clifford. Finding patterns in time series. In *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT, 1996.

[3] P. Bieganski, J. Riedl, and J. V. Carlis. Generalized suffix trees for biological sequence data: Applications and implementation. In *Proc. Hawaii Int'l Conf. on System Sciences*, 1994.

[4] T. Bozkaya, N. Yazdani, and M. Ozsoyoğlu. Matching and indexing sequences of different lengths. In *Proc. ACM CIKM*, 1997.

[5] W. W. Chu and K. Chiang. Abstraction of high level concepts from numerical values in databases. In *Proc. AAAI Workshop on Knowledge Discovery in Databases*, 1994.

[6] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *Proc. ACM SIGMOD*, 1994.

[7] D. Q. Goldin and P. C. Kanellakis. On similarity queries for time-series data: Constraint specification and implementation. In *Proc. Constraint Programming*, 1995.

[8] S. Park, W. W. Chu, J. Yoon, and C. Hsu. Efficient searches for similar subsequences of different lengths in sequence databases. In *Proc. IEEE ICDE*, 2000.

[9] S. Park, D. Lee, and W. W. Chu. Fast retrieval of similar subsequences in long sequence databases. Technical report, University of California, Los Angele, UCLA-CS-TR-990028, 1999.

[10] L. Rabinar and B.-H. Juang. *Fundamentals of Speech Recognition*. Prentice Hall, 1993.

[11] H. Shatkay and S. B. Zdonik. Approximate queries and representations for large data sequences. In *Proc. IEEE ICDE*, 1994.

[12] G. A. Stephen. *String Searching Algorithms*. World Scientific Publishing, 1994.

[13] J. T.-L. Wang, G.-W. Chirn, T. G. Marr, B. Shapiro, D. Shasha, and K. Zhang. Combinatorial pattern discovery for scientific data: Some preliminary results. In *ACM SIGMOD*, 1994.

[14] B.-K. Yi, H. V. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. In *IEEE ICDE*, 1998.