Fast Retrieval of Similar Subsequences in Long Sequence Databases

Sanghyun Park **Dongwon Lee** **Wesley W. Chu**

Department of Computer Science
University of California, Los Angeles
Los Angeles, CA 90095, USA

Email: {shpark,dongwon,wwc}@cs.ucla.edu

Last Revised: August 9, 1999

Abstract

Many Indexing techniques have been proposed to support the fast retrieval of similar sequences using the Euclidean distance metric. Since the Euclidean distance metric requires that two sequences be of the same length, it cannot be applied to sequences of different lengths. To remedy this problem, recent techniques (e.g, [5, 15]) have used the modified editing distance or the time warping distance, concentrating on the whole sequence matching. However, if their techniques are applied to the subsequences matching with different sequence lengths, the number of subsequences to be inspected during the search is quadratic to the average length $\bar{L}$ of data sequences, making the search algorithm suffer from severe performance degradation in long sequence databases.

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 linear to $\bar{L}$. In the aligned subsequence matching, sequences are segmented by piece-wise subsequences and only those subsequences starting and ending at segment boundaries are inspected at search time. We also present the indexing technique to support the fast retrieval of the similar aligned subsequences. Our indexing method is summarized as follows: First we extract the feature vector from each subsequence segment and group similar feature vectors together. Then, we convert each subsequence segment into the symbol of the corresponding group. Finally, from the sequences of symbols, we construct the generalized suffix tree (GST). At search time, the GST is traversed to find the subsequences whose lower-bound distances from a query sequence do not exceed the distance tolerance. The subsequences that are within the distance tolerance are obtained after discarding the false alarms. 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-time speed-up (653%).*

UCLA-CS-TR-990028

Contents

1 Introduction

Similarity searches in sequence databases plays an important role in many application domains such as information retrieval, data mining or clustering. Detecting stocks that have similar growth patterns or finding patients whose lung tumors have similar evolution characteristics are a few examples of such similarity searches. Although sequential scanning can be used for answering those queries, its processing time over large sequence databases is too costly.

Recently, several indexing techniques [1, 5, 8, 9, 15] have been proposed to speed up the processing of similarity searches. Researches on similarity searches can be classified by three criteria: 1) similarity measure (or distance function), 2) sequence representation, and 3) indexing method.

- The similarity measure defines the degree of proximity of two sequences. Many of the previous works [1, 8, 9] have used the Euclidean distance metric as the similarity measure. However, in applications where the lengths or the evolution rates of sequences may be different, other similarity measures, such as the modified editing distance [5] or the dynamic time warping distance [15], need to be used.

- Since even the simplest similarity measures are often too expensive to be applied on raw sequences, many previous works have taken the approach to extract the sets of features from sequences and to measure the similarity of two sequences using the distance between two features sets. For example, [8] used a few DFT coefficients of subsequences of fixed size, and [13] divided sequences into real-values functions and extract simple features from each segment. To guarantee no *false dismissals*, the similarity measure on feature sets should satisfy the so-called *lower-bounding lemma* [8].

- To speed up the similarity searches, the sets of features are managed by some indexing methods. [1, 8] used R* tree and [5] used vantage-point tree as index structures. It is known that all spatial access methods may generate false dismissals when the distance functions not satisfying *triangular inequality* are used as similarity measures of sequences [15]. The time warping distance and the modified editing distance based on proximity do not satisfy the triangular inequality, making spatial access methods unsuitable for indexing structures with these similarity measures.

On the other hand, as for the problems that these techniques aim to solve, most have so far focused on, to a greater or lesser extent, 1) same-length sequences, 2) whole sequence matching, or 3) relatively short sequences. Only recently a handful of new indexing techniques (e.g., [5, 8, 13, 15]) have been proposed to cope with the opposite situations having 1) different-length sequences, 2) subsequence matching, or 3) long[1] sequences. Although these techniques work well for each problem, when the problems are combined,

[1]Although it is difficult to define how much is *long* exactly, in this paper, we considered data sequences beyond 5,000

Notation	Description
N	number of elements in a sequence.
M	number of sequences in a database.
ϵ	distance tolerance given by a user.
$<>$	empty sequence.
$\vec{X}$	sequence 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. same as $\vec{X}$[p:$\|\vec{X}\|$].
$\vec{X}^S$	segmented sequence of $\vec{X}$.
$\vec{X}^F$	feature vector sequence of $\vec{X}$.
$\vec{X}^C$	categorized sequence of $\vec{X}$.
$\vec{X}^S$[p]	p-th subsequence segment of $\vec{X}^S$.
$\vec{X}^F$[p]	p-th feature vector of $\vec{X}^F$.
$\vec{X}^C$[p]	p-th symbol of $\vec{X}^C$.
$D_{sim}(\vec{X},\vec{Y})$	similarity measure between sequences $\vec{X}$ and $\vec{Y}$.
$D_{sim-lb1}(\vec{X},\vec{Y})$	first lower-bound similarity measure between sequences $\vec{X}$ and $\vec{Y}$.
$D_{sim-lb2}(\vec{X},\vec{Y})$	second lower-bound similarity measure between sequences $\vec{X}$ and $\vec{Y}$.

Table 1: List of notations.

they became insufficient. For instance, finding similar *subsequences* with *different lengths* is not a trivial problem; the number of subsequences to be inspected during the search is quadratic to the average length $\bar{L}$ of data sequences, making the search algorithm suffer from severe performance degradation in long sequence databases.

To remedy this problem, 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 linear to $\bar{L}$ of data sequences. In the aligned subsequence matching, sequences are segmented by piece-wise subsequences 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, and 3) the number of segments in $\vec{X}$[p:r] is the same as that of segments in the query sequence. There are several ways to obtain the piece-wise subsequence segments. Our proposal that does not require that the lengths of subsequence segments be same is elaborated in Section 4.

Using the aligned subsequence matching, our similarity measure between two sequences $\vec{X}$ and $\vec{Y}$ is defined as follows (Table 1 shows a list of notations used in this paper):

elements as long sequences. The data sequence of 5,000 elements can contain the daily ending prices of a stock for about twenty years.

$$D_{sim}(\vec{X},\vec{Y}) = \sum_{i=1}^{|\vec{X}^S|} D_{tw}(\vec{X}^S[i],\vec{Y}^S[i]) \tag{1.1}$$

where $\vec{X}^S[i]$ and $\vec{Y}^S[i]$ are the i-th subsequence segments of $\vec{X}$ and $\vec{Y}$, respectively, and $D_{tw}(\vec{X}^S[i],\vec{Y}^S[i])$ is the time warping distance between two segments. This formula can be rephrased as "the distance between two sequences is the sum of the time warping distances between each subsequence segment". Note that the number of subsequence segments of $\vec{X}$ and $\vec{Y}$ must be same.

Example 1: Suppose we want to find all the aligned subsequences of a sequence $\vec{X} = <1,3,6,4,3,2,1,0,1,3>$ that are similar to a query sequence $\vec{Q} = <3,1,0,1,3>$ within a distance tolerance $\epsilon = 2.5$. Further suppose that $\vec{X}$ and $\vec{Q}$ are segmented to $\vec{X}^S = <<1,3,6>,<4,3,2,1,0>,<1,3>>$ and $\vec{Q}^S = <<3,1,0>,<1,3>>$, respectively. Since $\vec{X}^S$ and $\vec{Q}^S$ have 3 and 2 subsequence segments, respectively, the segmented query sequence $\vec{Q}^S = <\vec{Q}^S[1],\vec{Q}^S[2]>$ can possibly be compared with only two combinations of subsequence segments of $\vec{X}^S$; $<\vec{X}^S[1],\vec{X}^S[2]>$ and $<\vec{X}^S[2],\vec{X}^S[3]>$. Then, the distances is computed as follows:

- $D_{sim}(<1,3,6,4,3,2,1,0>,<3,1,0,1,3>) = D_{tw}(\vec{X}^S[1],\vec{Q}^S[1]) + D_{tw}(\vec{X}^S[2],\vec{Q}^S[2])$
  $= D_{tw}(<1,3,6>,<3,1,0>) + D_{tw}(<4,3,2,1,0>,<1,3>) = 10 + 9 = 19.$
- $D_{sim}(<4,3,2,1,0,1,3>,<3,1,0,1,3>) = D_{tw}(\vec{X}^S[2],\vec{Q}^S[1]) + D_{tw}(\vec{X}^S[3],\vec{Q}^S[2])$
  $= D_{tw}(<4,3,2,1,0>,<3,1,0>) + D_{tw}(<1,3>,<1,3>) = 2 + 0 = 2.$

Since only $D_{sim}(<4,3,2,1,0,1,3>,<3,1,0,1,3>)$ has a distance within the given ϵ, the aligned subsequence that matches the query sequence $\vec{Q}$ is the $<4,3,2,1,0,1,3>$. ■

Although the aligned subsequence matching is significantly faster than the conventional subsequence matching (i.e., $O(M\bar{L}|\vec{Q}|)$ vs. $O(M\bar{L}^2|\vec{Q}|)$), we can still improve the search time further by adopting a sophisticated indexing method. That is, the search time can be decreased to $O(\frac{M\bar{L}|\vec{Q}|}{F})$, where F is some constant factor gained by the indexing. To achieve this gain, we used an indexing method constructed as follows. First, we extract the feature vector from each subsequence segment and group similar feature vectors together. Then, we convert each subsequence segment to a symbol of the corresponding group. Finally, from the sequences of symbols, we construct a generalized suffix tree (GST). At search time, the GST is traversed to find the subsequences whose lower-bound distances from a query sequence do not exceed the distance tolerance. The subsequences that are within the distance tolerance are obtained after discarding the false alarms.

The remaining paper is organized as follows. In Section 2, related works are surveyed. In Section 3, some preliminary information is introduced. Then, our proposed techniques are discussed in following sections; First, Section 4 describes our method to segment sequences to piece-wise subsequences. The indexing technique and its search algorithms are presented in Section 5 and in Section 6, respectively. Finally, our proposal is verified by the experimental results in Section 7 and some concluding remarks are followed in Section 8.

2 Related Works

Several approaches for fast retrieval of similar sequences have recently been 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 [8]. Assuming the minimum query length W is known in advance, features are extracted from every subsequence of size W and are mapped into multi-dimensional points. The mapped points are represented by their minimum bounding rectangles. Since both approaches of [1] and [8] use the Euclidean distance metric as similarity measures, sequences of different lengths or different evolution rates cannot be matched.

Sequence matching that allows transformations is proposed in [9, 11]. In [9], the sequences are grouped into equivalent classes according to shape-based transformations, and are represented by their normal forms from which the indexes are built. However, the normal forms do not handle the compressions or the stretches of element values along the time axis. The authors of [11] propose a class of transformations that can be used in a query language to express similarity with an R-tree index. Since an R-tree index is based on the triangular inequality, [11] may generate false dismissals with the time warping similarity measure.

The access methods of [5, 15] permit the matching of sequences of different lengths. [5] presents a modified version of edit distance, considering two sequences being matched if a majority of elements match. In [15], a time warping distance is used as a similarity measure with a two-step filtering process: a FastMap index filter followed by a lower-bound distance filter. Both approaches of [5, 15] focus on the whole sequence matching and use index structures based on the triangular inequality.

Similarity matching based on shapes of sequences is proposed in [2, 13]. [2] demonstrates a shape definition language (SDL) and provides an index structure for speeding up the execution of SDL queries. In [13], the authors introduce the notion of generalized approximate queries that specify the general shapes of data histories. Whereas both approaches of [2, 13] may handle the sequences of different lengths or different evolution rates, they cannot be used for applications that care about specific element values.

There are also several approaches for matching of biological sequences. [4] proposes to use a disk-based generalized suffix tree for solving the sequence alignment problem, and [14] addresses the problem of discovering patterns in protein databases with the similarity measure of a string edit distance. While we focus on the sequences of real numbers, the approaches of [4, 14] center on the sequences of characters. Furthermore, the algorithm of [14] uses a main-memory based generalized suffix tree, making it infeasible for a large sequence set.

3 Background

We assume that values of the sequences are real numbers. We denote a sequence $<x_1,\ldots,x_N>$ as $\vec{X}$. In what following subsections, we shall review some preliminary background to facilitate the discussion.

3.1 Time Warping

Time warping allows the sequence to be stretched or compressed along the time axis. To find the minimum difference between two sequences, time warping maps each element of a sequence to one or more neighboring elements of another sequence. For any non-null sequences $\vec{X}$ and $\vec{Y}$, the time warping distance is defined as follows[10]:

Definition 1: The time warping distance between sequences $\vec{X}$ and $\vec{Y}$ is:

$$
\begin{aligned}
D_{tw}(<>,<>) &= 0 \\
D_{tw}(\vec{X},<>) &= D_{tw}(<>,\vec{Y}) = \infty \\
D_{tw}(\vec{X},\vec{Y}) &= D_{base}(\vec{X}[1],\vec{Y}[1]) \;+\; min \begin{cases} D_{tw}(\vec{X},\vec{Y}[2:-]) \\ D_{tw}(\vec{X}[2:-],\vec{Y}[2:-]) \\ D_{tw}(\vec{X}[2:-],\vec{Y}[2:-])) \end{cases}
\end{aligned}
$$

where $D_{base}(\vec{X}[i],\vec{Y}[i]) = |\vec{X}[i]-\vec{Y}[i]|^2$. ■

$D_{tw}(\vec{X},\vec{Y})$ can be efficiently calculated using a dynamic programming technique [3] based on the recurrence relation $R_{tw}(i,j)$, where $1 \le i \le |\vec{X}|$ and $1 \le j \le |\vec{Y}|$.

Definition 2: The recurrence relation $R_{tw}(i,j)$ is:

$$
\begin{aligned}
R_{tw}(0,0) &= 0 \\
R_{tw}(i,0) &= R_{tw}(0,j) = \infty \\
R_{tw}(i,j) &= D_{base}(\vec{X}[i],\vec{Y}[i]) \;+\; min \begin{cases} R_{tw}(i,j-1) \\ R_{tw}(i-1,j) \\ R_{tw}(i-1,j-1) \end{cases}
\end{aligned}
$$

■

[2]Any distance metric in L_p is fine.

$R_{tw}(i, j)$ builds up the cumulative distance table as the computation proceeds. The final cumulative distance, $R_{tw}(|\vec{X}|, |\vec{Y}|)$ is the time warping distance between sequences $\vec{X}$ and $\vec{Y}$. The matching of elements can be traced backward in the table by choosing the previous cells with the lowest cumulative distance. For details, refer to [3].

3.2 Suffix Tree

A *trie* is an indexing structure used for indexing sets of keywords of varying sizes. A *suffix trie* [12] is a trie whose set of keywords comprises the suffixes of a single sequence. Nodes with a single outgoing edge can be collapsed, yielding the structure known as the suffix tree [12]. A *generalized suffix tree* (GST) [4, 14] is an extension of the suffix tree allowing for multiple sequences to be stored in the same tree. Each suffix of a sequence is represented by a leaf node. Precisely, $\vec{X}$[p:-] is expressed by a leaf node labeled with (id($\vec{X}$),p), where id($\vec{X}$) is an identifier of $\vec{X}$ 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:-]. The concatenation of the edge labels on the path from the root to the internal node, N_i, represents the longest common prefix of the suffixes represented by the leaf nodes under N_i. We use the notation $label(N_i, N_j)$ for the concatenated labels on the path from N_i to N_j. In what follows, we use the *trie* as a basis of our discussion although we used the *tree* in real implementation for its efficiency. This is because using the *trie* is simpler to explain than using the *tree*.

Example 2: Figure 1 illustrates the GST constructed from two sequences, $\vec{X}$ = <4,5,6,7,6,6> and $\vec{Y}$ = <4,6,7,8>, where \$ where \$ denotes an end marker of a suffix. ■

4 Segmentation

Ideally, the sequence representation should be succinct while retaining interesting features that the original sequence has. Similar sequence representations should correspond to similar original sequences and vice versa. To break up sequences, we use a divide & conquer algorithm shown in Algorithm 1 which is modified from the one in [13]. First, let us introduce the concept of the *peak* point.

Definition 3: Given three consecutive points ⟨ x, y, z ⟩ from a sequence $\vec{X}$, if 1) x ≤ y and y ≥ z or 2) x ≥ y and y ≤ z, then the point y is called the *peak* point. In an extreme case such as the formula y=3, every point becomes the peak point. In typical case, however, when monotonically increasing sequence is changed to monotonically decreasing or vice versa, the turning point becomes the peak point. ■

Using the peak point concept, our algorithm essentially does following: given a sequence, algorithm first scans the sequence and records all the peak points. Then, algorithm takes the line interpolating two end

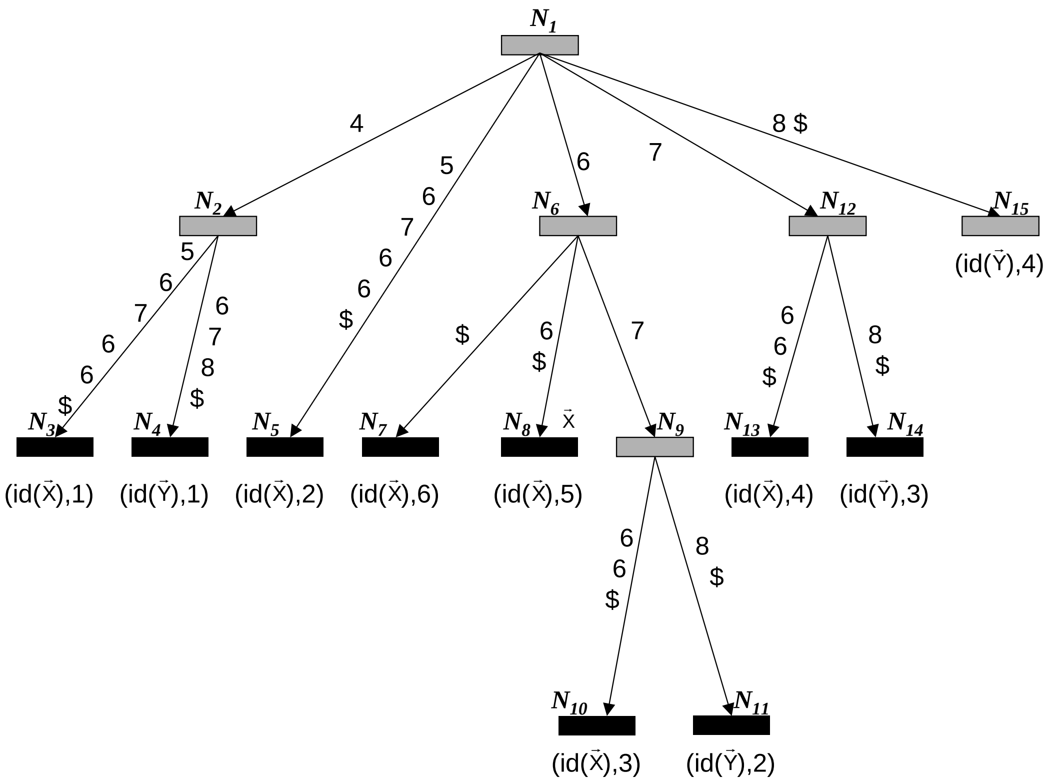

Figure 1: GST example.

points of the sequence, effectively breaks sequences at extremum points with maximum deviation, and recursively breaks those subsequences further until some threshold is exceeded. In addition, we add two improvements as follows:

- Algorithm in [13] scans whole N elements in a sequence $\vec{X}$ to find the extremum point in each recursion. Since this finding lasts for the number of peak points, say P, its worst case running time is $O(N * P)$. We observed that most of the time the extremum point is the peak point. Hence, our algorithm examines only peak points to find the extremum point in each recursion (line 2 of the Algorithm 1). The worst case running time of ours, then, is $O(N + P^2)$. When N is very large and $P \ll N$ (which is very likely the case in practice), ours is more efficient than the algorithm in [13].

- Undesirable subsequences can be filtered out as early as possible by the constraint check at line 6 of the Algorithm 1. This is desirable property when certain semantic constraints are identified early. For instance, in stock trading sequence data, if only subsequences whose increase or decrease rates are more than 5% are interested, then our algorithm can filter out those whose increase or decrease rates are less than 5% at early stage.

Note that the elements of $\vec{X}^S$ are subsequences instead of real numbers. Therefore, the number of elements in $\vec{X}^S$ is much smaller than that of $\vec{X}$. The *compaction ratio (C)* can be expressed as $C = \frac{|\vec{X}|}{|\vec{X}^S|}$.

Input : sequence $\vec{X} = <x_1,...,x_N>$, peak points $P = p_1,...,p_K$ where $p_i \in \vec{X}$ and $1 \leq \mathrm{K} \leq \mathrm{N}$, threshold t, constraint C

Output: segmented sequence $\vec{X}^S$

1 fit an interpolation line L to $\vec{X}$;
2 find max derivation point p_i from P and its value d;
3 remove p_i from P;

4 **if** $d < t$ **then**
    return $\vec{X}$;
else
5   split $\vec{X}$ into two subsequences α and β at P_i;
6   **if** *α or β violates C* **then**
7   throw away α and β;
8   **goto** line 2;
    else
9   call Segmentation(α, P, t, C);
10   call Segmentation(β, P, t, C);

Algorithm 1: Segmentation

C is also considered as the average number of elements in a subsequence segment.

Example 3: $\vec{X}$ = <4,5,8,8,8,8,9,11,8,4,3,7,10> is segmented to $\vec{X}^S$ where $\vec{X}^S[1]$ = <4,5,8,8,8,8,9,11>, $\vec{X}^S[2]$ = <8,4,3>, and $\vec{X}^S[3]$ = <7,10>. $|\vec{X}| = 13$ and $|\vec{X}^S| = 3$. Therefore, the compaction ratio $C = 13/3 = 4.3$. ■

5 Index Construction

To support the fast aligned subsequence matching, we need an efficient indexing scheme. First, we extract the feature vector from each subsequence segment created from the segmentation step and group similar feature vectors together using the clustering technique. Then, we convert each subsequence segment to a symbol of the corresponding group. Finally, from the sequences of symbols, we construct the GST.

5.1 Feature Extraction($\vec{X}^S \rightarrow \vec{X}^F$)

In this step, representative features are extracted from each subsequence segment. We use a 5 tuple feature vector $(L, V_1, V_L, \delta_+, \delta_-)$ for each subsequence segment α, where $L = |\alpha|$, $V_1 = \alpha[1]$, $V_L = \alpha[\mathrm{L}]$, and δ_+ and δ_- are the positive and negative maximum deviation from the interpolation line connecting the two points $(1,\ V_1)$ and $(L,\ V_L)$, respectively. Each feature shall be denoted using the dot notation

Feature Vector	L	V_1	V_L	δ_+	δ_-	interpolation line
α^F	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
δ^F	3	8	3	max{0,0,0} = 0	max{0,1.5,0} = 1.5	y = –2.5x + 10.5
γ^F	2	7	10	max{0,0} = 0	max{0,0} = 0	y = 3x + 4

Table 2: Subsequence feature vectors example.

like $\alpha.V_1$ or $\alpha.\delta_+$. A deviation value δ_+ can be computed by the Algorithm 2. (δ_- is a symmetric case of the δ_+ and omitted).

```
Input  : subsequence α[1 : L], Interpolate(i, L, V1, VL) = V1 + (VL − V1) ∗ (i − 1)/(L − 1)
Output: δ+

δ+ ← v ← 0;
for i ← 1 to L do
    v ← α[i] - Interpolate(i, L, V1, VL);          #compute deviation value.
    if v < 0 then v ← 0;                            #make v non-negative.
    if v > δ+ then δ+ ← v;                          #keep max value.
return δ+;
```

Algorithm 2: PositiveMaxDeviation.

Example 4: Consider three subsequences α = <4,5,8,8,8,8,9,11>, β = <8,4,3>, γ = <7,10>. Their corresponding feature vectors α^F, β^F, and γ^F are shown in Table 2. ■

5.2 Categorization($\vec{X}^F \rightarrow \vec{X}^C$)

In this step, feature vectors extracted in feature extraction step are mapped into corresponding symbols. Using the clustering technique (e.g, equal-length-interval, maximum-entropy, or MTAH [6])[3], we first group similar feature vectors together. Then, a unique symbol is assigned to each group and lower and upper bounds for feature vectors included in the group are kept in the mapping table. One such example is depicted in Table 3.

We use lb and ub to denote lower and upper bound of range, respectively. Using the dot notation, for instance, symbol A's feature δ_+ in Table 3 has range like $A.\delta_+.lb = 1.5$ and $A.\delta_+.ub = 2.5$. Then, according to the mapping table, sequences of feature vectors are converted to sequences of symbols. That is, given a 5-tuple feature vector α^F, if all features satisfy the corresponding ranges for the symbol A in the mapping table, then α^F is replaced to A. $\vec{X}^C$ is used to refer to the categorized sequence of $\vec{X}$. Note that elements of $\vec{X}^C$ are symbols such as A and the length of $\vec{X}^C$ is the same as that of $\vec{X}^S$. Consider the following illustrating example.

[3]For experimentation, the maximum-entropy is used.

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

Table 3: Mapping table example from 5-dimensional features to symbols.

Example 5: Consider $\vec{X}$= <4,5,8,8,8,8,9,11,8,4,3> is segmented into $\vec{X}^S$= <<4,5,8,8,8,8,9,11>, <8,4,3>>. After feature vectors are extracted from each subsequence, $\vec{X}^F[1]$ and $\vec{X}^F[2]$ are the same as α^F and β^F in Table 2, respectively. Further, by looking up Table 3, $\vec{X}^F[1]$ and $\vec{X}^F[2]$ can be mapped into symbols A and B, respectively. Therefore, $\vec{X}^C$= <A,B>. ■

5.3 GST Construction

After obtaining sequences of symbols, we use the GST as an index structure for fast subsequence matching. It has the following benefits:

- It is a good structure especially for subsequence matching since all possible suffixes of the given sequence is maintained in the GST.
- It is not based on triangular inequality so that it guarantees no false dismissals using the lower-bounding distance functions in index space.

There are several algorithms proposed to build the GST from sequences of symbols. We use an incremental disk-based GST construction method proposed in [4]. 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. A GST for a large set of sequences can be constructed by performing a series of binary merges of GSTs of increasing size. The merge operation of two GSTs has the advantage of supporting disk-based representations of GSTs in limited main memory. The construction of GST has the time complexity $O(Z)$ where Z is the total length of the sequences. In our case, Z is expressed as $Z = |\vec{X}_1^C| + ... + |\vec{X}_M^C|$.

6 Search Algorithm

Given the query sequence $\vec{Q}$ and the distance threshold ϵ, we want to find subsequences whose distances to $\vec{Q}$ are within ϵ. In this section, we propose a new search algorithm, SearchSubSequence(), that consists of three steps – two filterings and a post-processing - as shown in 2.

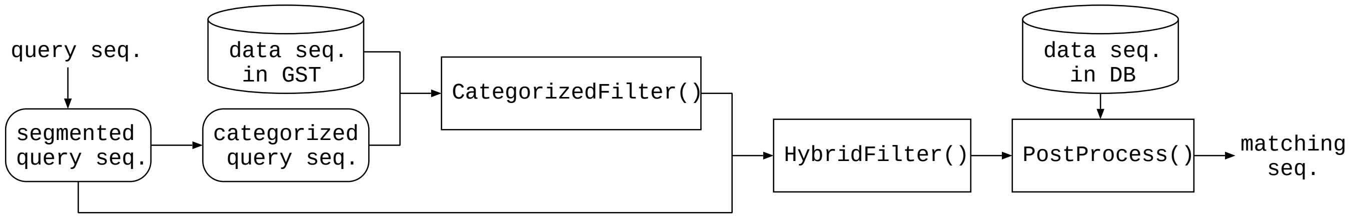

Figure 2: Overview of the SearchSubSequence() Algorithm.

1. **1st filtering**: During traversal of the GST, SearchSubSequence() computes the lower-bound distances between the *categorized* query sequence $\vec{Q}^C$ and the *categorized* subsequences $\vec{X}^C$ and finds all candidates whose distances are within ϵ. Since this is based on two *categorized* sequences, we call this step CategorizedFilter().

2. **2nd filtering**: SearchSubSequence() computes the lower-bound distances between the *segmented* query sequence $\vec{Q}^S$ and the *categorized* subsequences $\vec{X}^C$ belonging to the candidate set returned from CategorizedFilter() and finds all candidates whose distances are within ϵ. Since this is based on both *segmented* and *categorized* sequences, we call this step HybridFilter().

3. **post-processing**: the actual subsequences of the remaining candidates are retrieved and their distances from $\vec{Q}$ are computed and those whose distances are within ϵ are returned as the final answers. This step is called PostPorcess().

6.1 Search Algorithm: SearchSubSequence()

The SearchSubSequence() and CategorizedFilter() are shown in Algorithm 3 and 4. Algorithm HybridFilter() is identical to CategorizedFilter(), except that

- It has $\vec{Q}^S$ as an input instead of $\vec{Q}^C$.
- It uses different distance function, $D_{tw-lb2}(\text{label}(N, CN_i), \vec{Q}^S[1])$, at line 1 and
- Before traversing the GST, suffixes that are not contained in the candidate set identified by the CategorizedFilter() are pruned.

We assume that a function GetChildren(N) returns a list of children of the node N and GetRoot(N) returns the root node of N. Algorithm HybridFilter() and PostProcess() are omitted.

6.2 Converting Symbols to Ranges

Given a subsequence segment (and thus its feature vector), the corresponding symbol can be easily found by looking up the mapping table that stores the lower and upper bound values. However, going the other

Input : Root, $\vec{Q}$, ϵ
Output: answerSet

$\vec{Q}^S \leftarrow$ Segmentation($\vec{Q}$);
$\vec{Q}^C \leftarrow$ Categorization($\vec{Q}^S$);
candidateSet $\leftarrow$ answerSet $\leftarrow \emptyset$;

candidateSet $\leftarrow$ CategorizedFilter(Root, $\vec{Q}^C$, ϵ);
for $i \leftarrow$ *1* ***to*** $|candidateSet|$ **do**
    prune suffixes from GST not contained in candidateSet;
candidateSet $\leftarrow$ HybridFilter(Root, $\vec{Q}^S$, ϵ);
answerSet $\leftarrow$ PostProcess(candidateSet, $\vec{Q}^S$, ϵ);
return answerSet;

Algorithm 3: SearchSubSequence

Input : Node N, $\vec{Q}^C$, ϵ
Output: candidateSet

candidateSet $\leftarrow \emptyset$;
CN $\leftarrow$ GetChildren(N);

for $i \leftarrow$ *1* ***to*** $|CN|$ **do**
    dist $\leftarrow D_{tw-lb1}$(label(N,CN_i), $\vec{Q}^C[1]$);
    if $dist \leq \epsilon$ **then**
        if $|\vec{Q}^C| = 1$ **then** insert label(GetRoot(CN_i), CN_i) into candidateSet;
        else CategorizedFilter(CN_i, $\vec{Q}^C[2:-]$, $\epsilon-$dist);
return candidateSet;

Algorithm 4: CategorizedFilter

way around, given a symbol, it is difficult to find out all subsequence segments included in the symbol without scanning all sequences contained in a database or maintaining links from symbols to subsequence segments. However, using the lower and upper bound values of the symbol, we can infer the possible value ranges for each element position of subsequence segments included in the symbol. That is, given a symbol A, possible value range for A is $range(A) = <(lb_1, ub_1), ..., (lb_{A.L.ub}, ub_{A.L.ub})>$. lb_i and ub_i $(1 \leq i \leq A.L.ub)$ can be computed by the following formula (prefix A is omitted for briefness):

$$lb_i = \begin{cases} V_1.lb & \text{if i = 1} \\ \mathsf{Interpolate}(i, L.ub, V_1.lb, V_L.lb) - \delta_{-}.ub & \text{if 1 < i < L.ub} \\ V_L.lb & \text{if i = L.ub} \end{cases}$$

$$ub_h = \begin{cases} V_1.ub & \text{if h = 1} \\ \mathsf{Interpolate}(h, L.lb, V_1.lb, V_L.lb) + \delta_{+}.ub & \text{if 1 < h < L.lb} \\ max\{V_L.ub, \mathsf{Interpolate}(h, L.ub, V_1.ub, V_L.ub) + \delta_{+}.ub\} & \text{if h = L.lb} \\ \mathsf{Interpolate}(h, L.ub, V_1.ub, V_L.ub) + \delta_{+}.ub & \text{if L.lb < h < L.ub} \\ V_L.ub & \text{if i = L.ub} \end{cases}$$

Example 6: Let us compute $range(B)$ from the symbol B in Table 3. The maximum length of the $range(B)$ is 3 since L.ub = 3. Therefore, $range(B) = <(lb_1, ub_1), (lb_2, ub_2), (lb_3, ub_3)>$. 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. $lb_2 = \mathsf{Interpolate}(h, L.ub, V_1.lb, V_L.lb) - \delta_{-}$.ub = Interpolate(2,3,7,2) – 3 = 4.5 – 3 = 1.5. $ub_2 = max\{V_L.ub, \mathsf{Interpolate}(h, L.ub, V_1.ub, V_L.ub) + \delta_{+}$.ub} = Interpolate(2,3,9,4) + 1.5 = 6.5 + 1.5 = 8. Thus, $range(B)$ = <(7,9),(1.5,8),(2,4)>. This can be interpreted as "symbol B can have starting value between 7 and 9 and ending value between 2 and 4. Also it may have intermediate value between 1.5 and 8." ■

In the rest of the paper, we assume that there is a function Symbol2Range() which takes a symbol as input and returns the lower and upper bound of each element of the subsequence segments included in the symbol.

6.3 Similarity Measure of CategorizedFilter()

Since the similarity measure in Equation 1.1 is based on the distance between two *segmented* sequences, it is not directly applicable to the CategorizedFilter() whose distance function is based on two *categorized* sequences. To remedy this problem, we need to modify the similarity measure accordingly. Let us assume that $label(N, CN_i)$ is the symbol A and $\vec{Q}^C[1]$ is the symbol B. Further, assume that $\mathsf{Symbol2Range}(A)$ = RA and $\mathsf{Symbol2Range}(B)$ = RB. Then, the lower bound distance between two symbols is defined as follows:

$$D_{tw-lb1}(A, B) = min(D_{tw-lb1-sub}(RA[1:i], RB[1:j])) \tag{6.2}$$

where $A.L.lb \leq i \leq A.L.ub$ and $B.L.lb \leq j \leq B.L.ub$. Since $RA[1:i]$ and $RB[1:j]$ are also the sequences of ranges, in what follows, we use RA and RB notation instead. Then, $D_{tw-lb1-sub}(RA, RB)$ is defined further as follows:

$$\begin{aligned}
D_{tw-lb1-sub}(<>, <>) &= 0 \\
D_{tw-lb1-sub}(RA, <>) &= D_{tw-lb1-sub}(<>, RB) = \infty \\
D_{tw-lb1-sub}(RA, RB) &= D_{base-lb1}(RA[1], RB[1]) + min \begin{cases} D_{tw-lb1-sub}(RA, RB[2:-]) \\ D_{tw-lb1-sub}(RA[2:-], RB) \\ D_{tw-lb1-sub}(RA[2:-], RB[2:-]) \end{cases}
\end{aligned}$$

$$D_{base-lb1}(RA[x], RB[y]) = \begin{cases} 0 & \text{if RA[x] and RA[y] overlap} \\ RB[y].lb - RA[x].ub & \text{if RA[x].ub < RB[y].lb} \\ RA[x].lb - RB[y].ub & \text{if RA[x].lb > RB[y].ub} \end{cases}$$

$D_{tw-lb1-sub}(RA[1:i], RB[1:j])$ builds the distance table of size $i * j$. Therefore, $D_{tw-lb1}(A, B)$ requires the constructions of $(A.L.ub - A.L.lb + 1) * (B.L.ub - B.L.lb + 1)$ distance tables whose sizes vary from $A.L.lb * B.L.lb$ to $A.L.ub * B.L.ub$. Having defined the lower bound distance, now the actual similarity measure of the $\mathsf{CategorizedFilter}()$ is defined as follows:

$$D_{sim-lb1}(\vec{X}, \vec{Q}) = \sum_{i=1}^{|\vec{X}^C|} D_{tw-lb1}(\vec{X}^C[i], \vec{Q}^C[i]) \tag{6.3}$$

6.4 Similarity Measure of HybridFilter()

Again, since the similarity measure in Equation 1.1 is based on the distance between two *segmented* sequences, it is not directly applicable to the HybridFilter() whose distance function is based on both *segmented* and *categorized* sequences. Therefore, similar to CategorizedFilter(), we need to modify the similarity measure accordingly. Let us assume that $label(N, CN_i)$ is the symbol A, $\mathsf{Symbol2Range}(A)$ = RA and $\vec{Q}^S[1]$ is the segment β. Then, the lower bound distance between the symbol A and the segment β is defined as follows:

$$D_{tw-lb2}(A, \beta) = min(D_{tw-lb1-sub}(RA[1:i], \beta)) \tag{6.4}$$

where $A.L.lb \leq i \leq A.L.ub$. Further, $D_{tw-lb2-sub}()$ can be defined as follows:

$$\begin{array}{lcl}
D_{tw-lb2-sub}(<>,<>) & = & 0 \\
D_{tw-lb2-sub}(RA,<>) & = & D_{tw-lb1-sub}(<>,\beta) \;\; = \;\; \infty \\
D_{tw-lb2-sub}(RA,\beta) & = & D_{base-lb2}(RA[1],\beta[1]) \;\; + \;\; min \begin{cases} D_{tw-lb2-sub}(RA,\beta[2:-]) \\ D_{tw-lb2-sub}(RA[2:-],\beta) \\ D_{tw-lb2-sub}(RA[2:-],\beta[2:-]) \end{cases}
\end{array}$$

$$D_{base-lb2}(RA[x],\beta[y]) = \begin{cases} 0 & \text{if RA[x].lb} \leq \beta\text{[y]} \leq \text{RA[x].ub} \\ \beta[y] - RA[x].ub & \text{if RA[x].ub} < \beta\text{[y]} \\ RA[x].lb - \beta[y] & \text{if RA[x].lb} > \beta\text{[y]} \end{cases}$$

$D_{tw-lb2-sub}(RA[1:i],\beta)$ builds the distance table of size $i * |\beta|$. Therefore, $D_{tw-lb2}(A,B)$ requires the constructions of $(A.L.ub - A.L.lb + 1)$ distance tables whose sizes vary from $A.L.lb$ to $A.L.ub$. Having defined the lower bound distance, now the actual similarity measure of the HybridFilter() is defined as follows:

$$D_{sim-lb2}(\vec{X},\vec{Q}) = \sum_{i=1}^{|\vec{X}^C|} D_{tw-lb2}(\vec{X}^C[i],\vec{Q}^S[i]) \tag{6.5}$$

6.5 Lower Boundness

The similarity measures of CategorizedFilter() and HybridFilter() obey the lower-bounding lemma faithfully, leading our index structure to guarantee no false dismissals. Let us first describe the lower-boundness of $D_{tw-lb1}()$ and $D_{tw-lb2}()$ on which both $D_{sim-lb1}()$ and $D_{sim-lb2}()$ are based.

Theorem 1: For any two subsequence segments α and β, and their corresponding categorized symbols A and B, the following inequality holds:

$$D_{tw-lb1}(A,B) \leq D_{tw-lb2}(A,\beta) \leq D_{tw}(\alpha,\beta)$$

■

Proof: By the mappings from α to A, and β to B, we know that $A.L.lb \leq |\alpha| \leq A.L.ub$ and $B.L.lb \leq |\beta| \leq B.L.ub$. Assume that Symbol2Range(A) $= RA$ and Symbol2Range(B) $= RB$. Then, let us take the first $|\alpha|$ ranges from RA and first $|\beta|$ ranges from RB. By the definition of the Symbol2Range(), we get

$$RA[i].lb \leq \alpha[i] \leq RA[i].ub \qquad (1 \leq i \leq |\alpha|) \tag{6.6}$$

$$RB[j].lb \leq \beta[j] \leq RB[j].ub \qquad (1 \leq j \leq |\beta|) \tag{6.7}$$

By Equations 6.6 and 6.7 and definitions of $D_{base-lb1}()$, $D_{base-lb2}()$, and $D_{base}()$, we get

$$D_{tw-lb1-sub}(RA[1:1], RB[1:1]) \leq D_{tw-lb2-sub}(RA[1:1], \beta[1]) \leq D_{tw}(\alpha[1], \beta[1]) \tag{6.8}$$

Using the induction process from 6.8, we get

$$D_{tw-lb1-sub}(RA[1:i], RB[1:|\beta|]) \leq D_{tw-lb2-sub}(RA[1:i], \beta) \qquad (1 \leq i \leq A.L.ub) \tag{6.9}$$

$$D_{tw-lb2-sub}(RA[1:|\alpha|], \beta) \leq D_{tw}(\alpha, \beta) \tag{6.10}$$

By Equation 6.9 and definitions of $D_{tw-lb1}()$ and $D_{tw-lb2}()$, we get

$$D_{tw-lb1}(A, B) \leq D_{tw-lb2}(A, \beta) \tag{6.11}$$

In addition, by Equation 6.10 and definitions of $D_{tw-lb2}()$ and $D_{tw}()$, we get

$$D_{tw-lb2}(A, \beta) \leq D_{tw}(\alpha, \beta) \tag{6.12}$$

Finally, by Equations 6.11, and 6.12, Theorem 1 is TRUE. ■

Now, we show the lower-boundness of CategorizedFilter() and HybridFilter(). Note that CategorizedFilter() uses the similarity measure $D_{sim-lb1}()$ and HybridFilter() uses the similarity measure $D_{sim-lb2}()$.

Theorem 2: For any two sequences $\vec{X}$ and $\vec{Y}$, the following inequality holds:

$$D_{sim-lb1}(\vec{X}, \vec{Y}) \leq D_{sim-lb2}(\vec{X}, \vec{Y}) \leq D_{sim}(\vec{X}, \vec{Y})$$

■

Proof: By Theorem 1, we know that

$$D_{tw-lb1}(\vec{X}^C[i], \vec{Y}^C[i]) \leq D_{tw-lb2}(\vec{X}^C[i], \vec{Y}^S[i]) \leq D_{tw}(\vec{X}^S[i], \vec{Y}^S[i]) for \forall_i, 1 \leq i \leq |\vec{X}^S| \tag{6.13}$$

Remember that $|\vec{X}^S| = |\vec{X}^C| = |\vec{Y}^C| = |\vec{Y}^S|$. By Equation 6.13 and the definitions of $D_{sim-lb1}(\vec{X}, \vec{Y})$, $D_{sim-lb2}(\vec{X}, \vec{Y})$, and $D_{sim}(\vec{X}, \vec{Y})$, Theorem 2 is TRUE. ■

6.6 Analysis of the Algorithms

We use the notations in Table 4 for the analysis of the algorithms.

Notation	Description
$\bar{L}$	average length of data sequences.
C	average number of elements in subsequence segments (i.e., compaction ratio)
$\frac{\lvert\vec{Q}\rvert}{C}$	average number of subsequence segments in the query sequence.
$\frac{\bar{L}}{C}$	average number of subsequence segments in a data sequence.

Table 4: List of notations.

Complexity of Sequential Scanning: The complexity for computing the time warping distance between two subsequence segments is $O(C^2)$. The complexity for measuring the modified time-warping distance between the segmented query sequence and the aligned subsequence is $O(\frac{C^2|\vec{Q}|}{C}) = O(C|\vec{Q}|)$. The average number of the aligned subsequences with $\frac{|\vec{Q}|}{C}$ subsequence segments in a data sequence is $(\frac{\bar{L}}{C} - \frac{|\vec{Q}|}{C} + 1)$. Then, 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 $O(M|\vec{Q}|\bar{L})$.

Complexity of $\mathsf{CategorizedFilter()}$: The complexity for computing $D_{tw-lb2}()$ is the same as $D_{tw}()$, but is reduced to $O(1)$ if we pre-compute all the distances between symbols and keep them in table. The complexity of $\mathsf{CategorizedFilter}$ is $O(\frac{M|\vec{Q}|(\bar{L}-|\vec{q}|+C)}{C^2R_1} + n_1C|\vec{Q}|)$ where R_1 (≥ 1) is the reduction factor saved by sharing edges in the GST and n_1 is the number of aligned subsequences requiring the post-processing. If $\bar{L} \gg |\vec{Q}|$, the complexity becomes $O(\frac{M\bar{L}|\vec{Q}|}{C^2R_1} + n_1C|\vec{Q}|)$.

Complexity of $\mathsf{HybridFilter()}$: The complexity for computing $D_{tw-lb2}()$ is the same as $D_{tw}()$. The complexity of $\mathsf{HybridFilter}()$ is $O(\frac{M|\vec{Q}|(\bar{L}-|\vec{Q}|+C)}{R_2} + n_2C|\vec{Q}|)$ where R_2 ($>> 1$) is the reduction factor saved by sharing edges in the GST and n_2 is the number of aligned subsequences requiring the post-processing. Similar to the $\mathsf{CategorizedFilter}()$, if $\bar{L} \gg |\vec{Q}|$, the complexity becomes $O(\frac{M\bar{L}|\vec{Q}|}{R_2} + n_2C|\vec{Q}|)$.

7 Experimental Results

We used the set of data sequences from UC Irvine KDD Archive (http://kdd.ics.uci.edu) to test the effectiveness of our approach. The dataset, called "Pseudo Periodic Synthetic Time Series", is specially designed for testing indexing schemes in time series databases. The actual sequence is generated by the following function:

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

where $0 \leq \vec{t} \leq 1$. From the ten 100,000-element data sequences, we created 90 sequences (each has 10,000-elements) out of the original 9 sequences and randomly extracted query shapes from the 10-th sequence.

Figure 3 shows the results of the experimentation. Our scheme consistently outperformed the sequential scanning and achieved up to 6.5-time speed-up (653%).

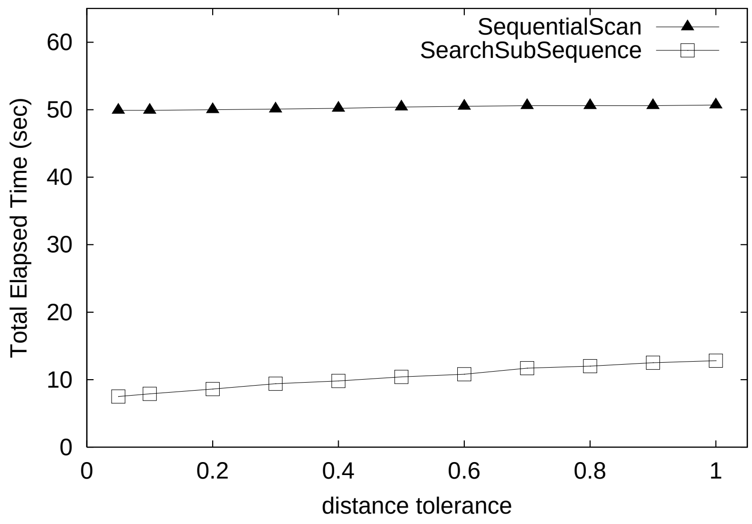

Figure 3: Performance comparison between SequentialScan and SearchSubSequence algorithms

8 Conclusion

In this paper, we proposed a novel sequence matching scheme, called the aligned subsequence matching, that has a time complexity $O(M\bar{L}|\vec{Q}|)$ for the retrieval of similar subsequences of different lengths. Our scheme is useful in applications handling long sequence databases. To speed up the aligned subsequence matching, we also presented an efficient indexing method that is based on the generalized suffix tree (GST) and two lower-bounding distance functions. Unlike R* tree and vantage-point-tree, the GST does not assume the triangular inequality, leading our indexing structure to guarantee no false dismissals. The experiments on a synthetic dataset demonstrated the effectiveness of our proposed approach.

References

[1] R. Agrawal, C. Faloutsos, A. Swami, "Efficient Similarity Search in Sequence Databases", *Proc. FODO*, Evanston, IL, 1993.

[2] R. Agrawal, G. Psaila, E. L. Wimmers, M. Zat, "Querying Shapes of Histories", *Proc. VLDB*, Zurich, Switzerland, 1995.

[3] D. J. Berndt, J. Clifford, "Finding Patterns in Time Series", *Advances in Knowledge Discovery and Data Mining, AAAI/MIT*, 1996.

[4] P. Bieganski, J. Riedl, J. V. Carlis, “Generalized Suffix Trees for Biological Sequence Data: Applications and Implementation”, *Proc. Hawaii Int'l Conf. on System Sciences*, 1994.

[5] T. Bozkaya, N. Yazdani, M. Özsoyoğlu, “Matching and Indexing Sequences of Different Lengths”, *Proc. ACM CIKM*, Las Vegas, NV, 1997.

[6] W. W. Chu, H. Yang, et el., “CoBase: A Scalable and Extensible Cooperative Information System”, *JIIS*, 1996.

[7] C. Faloutsos, K. Lin, “FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets”, *Proc. ACM SIGMOD*, San Jose, CA, 1995.

[8] C. Faloutsos, M. Ranganathan, Y. Manolopoulos. “Fast Subsequence Matching in Time-Series Databases”, *Proc. ACM SIGMOD*, Minneapolis, MN, 1994.

[9] D. Q. Goldin, P. C. Kanellakis, “On Similarity Queries for Time-Series Data: Constraint Specification and Implementation”, *Proc. Constraint Programming*, Marseilles, 1995.

[10] L. Rabiner, B.-H. Juang, “Fundamentals of Speech Recognition”. *Prentice Hall*, 1993.

[11] D. Rafiei, A. Mendelzon, “Similarity-based Queries for Time Series Data”, *Proc. ACM SIGMOD*, Tucson, AZ, 1997.

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

[13] H. Shatkay, S. B. Zdonik, “Approximate Queries and Representations for Large Data Sequences”, *Proc. IEEE ICDE*, 1994.

[14] J. T.-L. Wang, G.-W. Chirn, T. G. Marr, B. Shapiro, D. Shasha, K. Zhang, “Combinatorial Pattern Discovery for Scientific Data: Some Preliminary Results”, *Proc. ACM SIGMOD*, Minneapolis, MN, 1994

[15] B.-K. Yi, H. V. Jagadish, C. Faloutsos, “Efficient Retrieval of Similar Time Sequences Under Time Warping”, *Proc. IEEE ICDE*, 1998.