A Suffix Tree for Fast Similarity Searches of Time-Warped Sub-Sequences in Sequence Databases

SangHyun Park [*], Wesley W. Chu[*] , Jeehee Yoon [+], Chihcheng Hsu [#]

[*] Department of Computer Science, University of California, Los Angeles
[+] Department of Computer Engineering, Hallym University
[#] Santa Teresa Lab., IBM

Abstract

Several indexing techniques have been proposed to process similarity queries in sequence databases. Most of them focus on finding similar sequences of the same length using the Euclidean distance metric. However, in some applications where the elements of sequences may be sampled at different rates, the time warping distance is a more suitable similarity measure. In this paper, we propose an indexing technique based on a suffix tree for fast retrieval of similar sub-sequences under time warping. The search algorithm for a suffix tree is extended to provide similarity searches, and the concept of categorization is applied to reduce index size and to accelerate query processing. A greater reduction of index size is achieved using a sparse suffix tree and more speed-up is attained by the fast estimation of the time warping distances between non-stored suffixes and a query sequence. Our method guarantees no false dismissals since the actual time warping distances are always lower-bound in the index space. Our access method can also be used to answer shape-based queries since approximate shapes of sub-sequences are maintained in the index space. Experiments on stock and artificial sequences show that our approach is about 4 times faster than sequential scanning with a relatively small index space, and the performance gains increase up to 20 times as the size of indexes grows.

1. INTRODUCTION

Similarity searches in sequence databases are important in many application domains, such as information retrieval, data mining and clustering. Detecting stocks that have similar growth patterns and finding patients whose lung tumors have similar evolution fashions are a few examples of similarity queries. Sequential scanning is the simple method for answering those queries, but it may require an enormous processing time over large sequence databases. Recently, several indexing techniques [1,2,4,10] have been proposed to process similarity queries more quickly.

Most of the previous techniques [1,2,4] on similarity searches use the Euclidean distance metric as a similarity measure and apply the following ideas: extract features from sequences, map them into multi-dimensional points and let spatial access methods manage the mapped points. This idea can be extended to similarity matching of sub-sequences. However, it is not feasible to build spatial indexes from all sub-sequences since there are O(M*N*N) sub-sequences in M sequences with an average length N. This problem is addressed in [2] assuming knowledge of the minimum query length.

In some applications, like the matching of voice, audio, and medical signals, where the elements of sequences may be sampled at different rates, the time warping distance [6,11] is a more suitable similarity measure than the Euclidean distance metric. Under time warping, one element of a sequence can be matched to one or more elements of another sequence. So, two matching sequences are not required to have the same length. As an example [7], let us consider two sequences S_i = <20, 20, 21, 21, 20, 20, 23, 23> and S_j = <20, 21, 20, 23> where the sequence S_i is the closing price of a stock taken every day and S_j is the closing price of another stock taken every other day. S_i and S_j cannot be compared directly because the sequence S_i is longer than S_j. The Euclidean distance between S_j and any sub-sequence of length four of S_i is greater than 1.41. However, if we duplicate every value of the sequence S_j using a time warp, we find that the two sequences are identical.

In matching similar sequences, it is important to prevent the occurrence of false dismissals. It is said that false dismissals occurr when a sequence similar to a query sequence is not included in an answer-set. One important property of a time warping distance is that it does not satisfy the triangle inequality [4]. If a time warping distance is used as a similarity measure, all the spatial access methods as well as all the methods using distance/metric/vantage-point trees cannot avoid false dismissals. This is based on the fact that any indexing technique assuming the triangle inequality implicitly or explicitly cannot avoid producing false dismissals when the distance function dissatisfying the triangle inequality is used as a similarity measure [4].

In some other applications, the shapes of the sub-sequences are more important than the actual element values. The shapes are preserved even if every element value is scaled or shifted by a same amount. Finding sub-sequences that have monotonically increasing patterns and finding sub-sequences that show “goalpost fever” patterns are two examples of shape-based queries [9,10]. “Goalpost fever” [9] is one of the symptoms of Hodgkin’s disease that behaves as two consecutive fevers during a 24 hour period. It is not easy to answer shape-based queries if the access methods do not preserve the information about the shapes of the sequences.

In this paper, we propose a new indexing technique, which guarantees no false dismissals, for fast similarity retrieval of time-warped sub-sequences. A suffix tree [12] is used as an index structure and its search algorithm is extended to support similarity searches. The concept of categorization is applied to reduce the index size. Using categorization, we first convert sequences into their categorized representations and then construct a categorized suffix tree from the converted sequences. To prevent the occurrence of false dismissals under time warping, we define the distance function $D_{lb\text{-}warp}()$ for computing the lower-bound time warping distances between categorized sequences and a query sequence. We further reduce the size of a suffix tree using a sparse suffix tree [13] that stores only a subset of suffixes whose first values are different from their immediate preceding elements. During query processing, the non-stored suffixes are detected from the stored suffixes and their lower-bound time warping distances from a query sequence are calculated quickly by $D'_{lb\text{-}warp}()$. As the lower-bound time warping distance functions are used in the index space, the sequences dissimilar to a query sequence may not be filtered out. These sequences are called *false alarms*. False alarms are detected and discarded during post-processing.

Our access method may also be used to answer shape-based queries since the approximate shapes of sub-sequences are maintained in the index space. During query processing, we can trace the changing patterns of categorized values to locate the sub-sequences having similar shapes to a given query shape. The step for detecting and discarding false alarms is also needed since only approximate shapes are maintained in the index space.

This paper is organized as follows. Background and related works are described in section 2. In section 3, the method to use a suffix tree for similarity search is illustrated. Section 4 presents categorization techniques to reduce index size and to speed-up query processing. Experimental results are given in section 5.

2. BACKGROUND AND RELATED WORKS

Let us first describe the basics of time warping and a suffix tree. Table 1 lists some symbols used in this paper.

Symbols	Definitions
<>	empty sequence
$Len(S_i)$	length of the sequence S_i
$S_i[p]$	p^{th} element of the sequence S_i
$S_i[p:q]$	sub-sequence of S_i, including elements in positions p through q
$S_i[p:-]$	sub-sequence of S_i, including elements in positions p through the end

Table 1: Symbols used in this paper

2.1 Time warping

Time warping allows one element of a sequence to be matched to one or more elements of a target sequence with restrictions [11] like monotonicity, continuity, boundary conditions, and warping window. The time warping distance is suitable for applications, like the matching of voice, audio and medical signals (electrocardiograms), where the elements of sequences may be sampled at different rates.

Definition 1 Given any two non-null sequences S_i and S_j, the time warping distance, $D_{warp}()$, is defined as follows [6].

$$
\begin{aligned}
D_{warp}(<>, <>) &= 0. \\
D_{warp}(S_i, <>) &= D_{warp}(<>, S_j) = \infty \\
D_{warp}(S_i, S_j) &= D_{base}(S_i[1], S_j[1]) + \min (D_{warp}(S_i, S_j[2:-]) + D_{warp}(S_i[2:-], S_j) + D_{warp}(S_i[2:-], S_j[2:-])) \\
D_{base}(S_i[1], S_j[1]) &= | S_i[1] - S_j[1] |
\end{aligned}
$$

$D_{base}()$ on two numeric values can be any of the distance functions, but we assume that it is defined as the city-block distance. $D_{warp}(S_i, S_j)$ can be calculated efficiently by the dynamic programming technique based on the recurrence relation $\gamma(x, y)$.

Definition 2 Given any two non-null sequences S_i and S_j, the recurrence relation $\gamma(x, y)$ ($x = 1,2,\ldots,Len(S_i)$, $y = 1,2,\ldots,Len(S_j)$) which calculates the cumulative time warping distances between elements of S_i and S_j is defined as follows [11].

$$
\begin{aligned}
&\gamma(0, 0) = 0 \\
&\gamma(x, 0) = \gamma(0, y) = \infty \\
&\gamma(x, y) = D_{base}(S_i[x], S_j[y]) + \min (\gamma(x, y-1), \gamma(x-1,y), \gamma(x-1,y-1))
\end{aligned}
$$

The dynamic programming algorithm [11] fills in the table of cumulative distances as the computation proceeds. The final cumulative distance, $\gamma(Len(S_i), Len(S_j))$, is the desired distance between S_i and S_j, and the minimum matching can be traced backward in the table – choosing the previous cells with the lowest cumulative distance. This computation has the complexity $O(Len(S_i) * Len(S_j))$.

2.2 Suffix Tree

A suffix tree [12] is an index structure that has been proposed as a fast access method to locate sub-strings (or sub-sequences) that exactly match a query string (or a query sequence). A suffix is a sub-sequence that ends with the last element of a sequence, and a prefix is a sub-sequence that starts with the first element of a sequence. In a sequence whose length is N, there are N suffixes and N prefixes.

Like any other tree, a suffix tree consists of nodes and edges. The leaf nodes are labeled with the identifiers of the sequences and the starting positions of the suffixes that they represent. The internal nodes N_i of the tree are of degree ≥ 2 (with the exception of the root of a trivial suffix-tree), and represent the longest common prefixes of the suffixes represented by the leaf nodes under N_i. The sub-sequences represented by any node of the tree may be obtained by concatenating labels associated with the edges on the unique path from the root to the particular node in question. Therefore, the suffixes are obtained by concatenating the labels associated with the edges on the paths from root to leaf. We use the symbol Edge(N_i, N_j) for the edge connecting the node N_i(parent) to the node N_j(child). A suffix tree can be constructed with complexity $O(M * N)$, where M is the number of sequences and N is the average length of the sequences. The total number of nodes in the tree is constrained due to two facts: there are $O(M * N)$ leaf nodes; and the degree of any internal node is at least 2. Therefore, the maximum number of nodes and overall space requirement of the suffix tree is linear in $M * N$ [12].

Example 1 We consider two sequences S_1 = <4, 5, 6, 7, 6, 6> and S_2 = <4, 6, 7, 8>. The constructed suffix tree is shown in Figure 1. We use the symbol '\$' as the end mark of the suffixes.

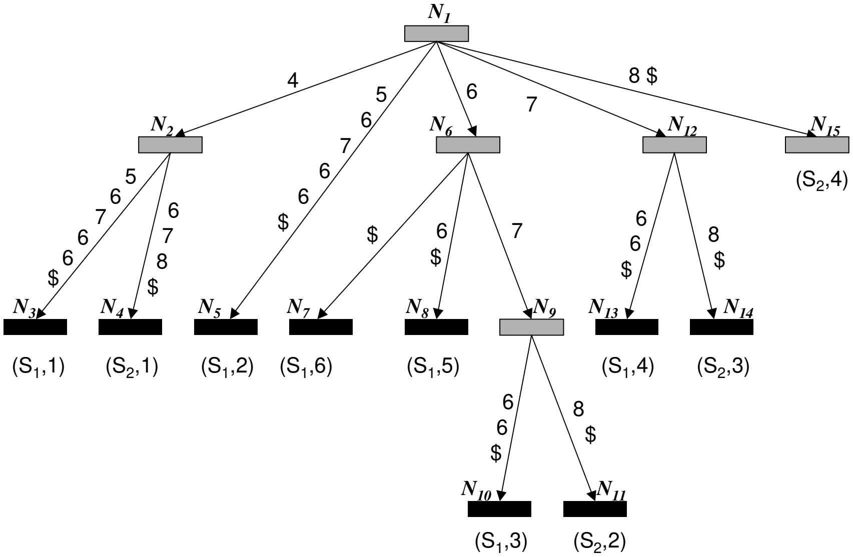

Figure 1: Suffix Tree from S_1=<4, 5, 6, 7, 6, 6> and S_2=<4, 6, 7, 8>

The algorithm to locate sub-sequences that exactly match a query sequence Q is given in Figure 2. This algorithm starts from the root and quickly progresses through the tree with complexity O(Len(Q)).

```
Algorithm: ExactSearch(CurNode)
Begin
❑ Visit the node, CurNode.
❑ Select the child node, ChildNode, whose associated label matches a prefix of Q.
❑ Remove the matched prefix from Q.
❑ If Q becomes empty,
    then report all leaf nodes under ChildNode as answers,
    else call ExactSearch(ChildNode).
End
```

Figure 2: Exact Search Algorithm for a suffix tree

2.3 Related Works

Several approaches for fast retrieval of similar sequences have recently been proposed. In [1], sequences of the time domain are converted into sequences of the frequency domain by the Discrete Fourier Transform and are subsequently mapped into multi-dimensional points that are managed by the R^*-tree. In [2], the technique of [1] is extended to locate similar sub-sequences. Assuming the minimum query length *W* is known in advance, features are extracted from every sub-sequence of size *W* and are mapped into multi-dimensional points. The mapped points are represented by their minimum bounding rectangles. Because the approaches of [1] and [2] do not permit the time warping of elements, the sequences with different sampling rates can not be matched.

Sequence matching that allows transformations of sequences are proposed in [5] and [7]. In [5], the sequences are grouped into equivalent classes according to shape-based transformations such as scaling and shifting, and are represented by their normal forms from which the indexes are built. However, the normal forms do not consider the time warping of elements. In [7], authors proposed a class of transformations that can be used in a query language to express similarity. The transformations include moving average, time warping and reversing. They implement similarity matching under those transformations on top of an R-tree index. Since the R-tree index is based on triangular inequality, they may generate false dismissals under time warping.

The access methods of [4] and [10] permit the matching of sequences of different lengths. In [10], they use a modified version of edit distance and consider two sequences matching if a majority of elements in the sequences match. For efficient retrieval of matching sequences, they first group data sequences by length and then index the groups by vp-trees (vantage point trees). However, they may generate false dismissals under a non-metric edit distance function. In [4], they use a time warping distance as a similarity measure. The filtering process consists of two steps: FastMap index filter and lower-bounding distance filter. Lower-bounding distance filter is used to quickly discard many false alarms that FastMap [3] introduces. Note that their approach is also based on triangular inequality. So, they can not guarantee no false dismissals.

Similarity matching based on shapes of sequences is proposed in [8] and [9]. In [9], they present a shape definition language, called SDL for retrieving sequences based on their shapes. SDL is able to perform a "blurry" match where the user cares about the overall shapes but does not care about the specific values. They provide an index structure for speeding up the execution of SDL queries. In [8], the authors introduce the notion of generalized approximate queries that specify the general shapes of data histories without depending on specific values. To support those queries, they break sequences into meaningful sub-sequences and represent them as mathematical functions.

3. SIMILARITY SEARCH ON A SUFFIX TREE

The problem we are trying to solve is formally defined as: *Given M sequences S_1, S_2, …, S_M of arbitrary lengths, a query sequence Q and a user given threshold ε, we want to find sub-sequences $S_i[p:q]$ (i = 1, 2, … M) whose time warping distances from Q are less than or equal to ε.*

Our proposed solution to the above problem uses a suffix tree as an index structure. We construct a suffix tree from the suffixes of all data sequences in databases using the classical construction algorithm [12]. Our similarity search algorithm for finding time-warped sub-sequences is based on the exact search algorithm defined in Figure 2. However, the exact search algorithm can not be directly applied to our problem domain since it is based on exact matching and does not allow the time warping of elements. The modified search algorithm is given in Figure 3. From the root, it traverses the suffix tree using a depth-first downward traversal approach. When it visits a node, it inspects each child node to find new answers and to determine whether further going-down is needed. This inspection process consists of building and checking the cumulative distance table.

```
Algorithm SimilaritySearch-on-SuffixTree(CurNode, CurTable)
Begin
❑ Visit the node, CurNode.
❑ For each child node, ChildNode, of CurNode, do the following.
    ▪ Using the recurrence relation $\gamma(x, y)$, construct a new cumulative distance table, NewTable, on top of CurTable
      for a query sequence and the label associated with the edge, Edge(CurNode, ChildNode).
    ▪ Find new answers by inspecting the last columns of NewTable and insert them into the answer-set.
    ▪ Determine whether we need to go further down by checking all columns of the last row of NewTable.
    ▪ If further visit is required, call SimilaritySearch-on-SuffixTree (ChildNode, NewTable).
End
```

Figure 3: Similarity search algorithm for a suffix tree

Let us assume that the search algorithm visits the node, N_i. The first step is to build a cumulative distance table for a query sequence Q and the label associated with the edge pointing to each child node of N_i. If N_i is a root node, the cumulative distance table is built from the bottom. Otherwise, it is built by augmenting new rows on the current cumulative distance table that has been accumulated from the root to N_i. As the labels associated with edges are located on Y-axis and a query sequence Q on X-axis, the cumulative distance table becomes taller as the searches proceed to leaf nodes.

The next step is to examine the last columns of newly added rows of the cumulative distance table to find new answers. If the last column of the k^{th} row has a value less than or equal to the user-given threshold ε, the prefix of length k from the label of Y-axis is inserted into the answer-set. The final step is to check all columns of the last row to determine whether or not further going-down is needed. If at least one column of the last row has a value less than or equal to ε, we continue down the tree to find more answers. Otherwise, the search moves to the next child node of N_i. This branch-pruning process is based on Theorem 1.

Theorem 1 If all columns of the last row of the cumulative distance table have values greater than a user-given threshold ε, adding more rows to this table does not yield any new answers.

Proof The proof is shown in appendix A.

Example 2 For a suffix tree shown in Figure 1, a query sequence $Q = \langle 3, 4, 4 \rangle$ and $\varepsilon = 2$, we first visit the root node N_1 and check each child node of N_1. To inspect the node N_2, we build the cumulative distance table having one row by locating the label of Edge(N_1, N_2) on Y-axis and Q on X-axis, and filling the cells of the row with the cumulative time warping distances. As the last column of the first row has the value 1, which is smaller than ε, the label *<4>* of Y-axis is inserted into the answer-set. The search continues to check the node N_3 since there are columns of the first row having the values not greater ε. The five new elements corresponding to the label of Edge(N_2, N_3) are located on top of the first element of Y-axis, and then five new rows are filled with the cumulative time warping distances. As the last column of the second row has the value 2, which is equal to ε, *<4,5>* is inserted into the answer-set. Even if there are children nodes under N_3, the search does not continue down since all columns of the last row have the values larger than ε. Thus, the search continues to check the node N_4. Table 2 shows the cumulative distance table when the search algorithm checks the node N_3.

row 6→	6	16	11	11
row 5→	6	13	9	9
row 4→	7	10	7	7
row 3→	6	6	4	4
row 2→	5	3	2	*2*
row 1→	4	1	1	*1*
		3	4	4

Table 2: Cumulative distance table for a query sequence Q=<3, 4, 4> and the label <4, 5, 6, 7, 6, 6> on the path from N_1 to N_3 of the suffix tree of Figure 1

The similarity search algorithm defined in Figure 3 executes faster than sequential scanning due to the shared cumulative distance tables and the branch-pruning process. This performance increases as the number of common subsequences grows.

4. CATEGORIZATION

In this section, we shall introduce the concept of categorization as a means of reducing the index size and accelerating query processing. To categorize element values, we divide the ranges of element values into sub-ranges. Each subrange is represented by a simple identifier called *the category-id*, and each element value is represented by its corresponding category-id. Thus, the sequences of values are converted into the sequences of category-ids.

4.1 Categorization Methods

We shall now introduce three categorization methods.

4.1.1 Equal-Length-Interval Categorization

As the name implies, all the categories have equal interval length $(MAX\text{-}MIN) / N_c$ where MIN is a minimum value of sequences, and MAX is a maximum value of sequences, and N_c is the number of categories. This categorization is simple and fast, but it loses information on the sequences because it ignores value or frequency distribution of the sequences.

4.1.2 Maximum-Entropy Categorization

The entropy [14] of categorization is defined as: $H(C) = \sum_{i=1}^{Nc} P(C_i) \log p(C_i)$ where $P(C_i)$ is the probability that an element is included in the i^{th} category. To minimize the loss of information for the sequences, maximum-entropy categorization decides the boundaries of categories that generate maximum entropy value. The boundaries can be determined easily by making all categories include the same number of elements ($P(C_1) = P(C_2) = \ldots = P(C_{Nc})$).

4.1.3 Minimum-Cut Categorization

Given two categories C_{i-1} and C_i, if two values $<V_{j-1}, V_j>$ satisfies the condition $V_{j-1} \in C_{i-1}$ and $V_j \in C_i$, then we say the values are cut by the boundary of C_{i-1} and C_i. Then, total number of cuts of the categorization C is defined as follows. $T(C) = \sum_{i=2}^{Nc} CUTS\ (C_{i-1}, C_i)$ where $CUTS(C_{i-1}, C_i)$ is the number of cuts made by the boundary between category C_{i-1} and category C_i. To maximize the total number of common sub-sequences, this method determines the category boundaries that generate the minimum number of total cuts.

4.2 Suffix Tree with Categorization

After categorizing the element values of the sequences, we convert the sequence of numbers into a sequence of category-ids. We can then build a suffix tree based on the categorized sequences. We call this suffix tree *the categorized suffix tree (CST)*. The categorized suffix tree is constructed using the same construction algorithm of the suffix tree but the edges now represent the categorized sub-sequences. In general, a categorized suffix tree has more common edges than an original suffix tree; hence the tree is smaller and the query processing is faster.

Example 3 Three categories are produced from $S_1 = <4, 5, 6, 7, 6, 6>$ and $S_2 = <4, 6, 7, 8>$ using the maximum-entropy categorization. The range of each category is shown in Table 3. According to this categorization, S_1 and S_2 are converted to their categorized representations, $CS_1 = <C_1, C_1, C_2, C_3, C_2, C_2>$ and $CS_2 = <C_1, C_2, C_3, C_3>$. Note that CS_1 and CS_2 have more common sub-sequences than those included in S_1 and S_2.

Category	MIN	MAX
C_1	4	5
C_2	6	6
C_3	7	8

Table 3: Minimum and maximum value of each category produced by maximum-entropy categorization from $S_1 = <4,5,6,7,6,6>$ and $S_2 = <4,6,7,8>$

The similarity search algorithm defined in Figure 3 needs to be modified to reflect the categorized representation of sequences. First, the recurrence relation $\gamma(x, y)$ is changed to the lower-bound recurrence relation $\gamma_{lb}(x, y)$ to construct a lower-bound cumulative distance table for a query sequence and a categorized sub-sequence. And, post-processing is added at the final stage to discard false alarms. Note that the actual element values of the query sequence is used to compute $\gamma_{lb}()$. The lower-bound time warping distance function $D_{lb\text{-}warp}()$ and its corresponding lower-bound recurrence relation $\gamma_{lb}()$ are defined as follows.

Definition 3 Given any two non-null sequences S_i and S_j, the distance function $D_{lb\text{-}warp}()$ that returns the lower-bound time warping distance between S_i and S_j is defined as follows.

$$
\begin{aligned}
D_{lb\text{-}warp}(<>, <>) &= 0. \\
D_{lb\text{-}warp}(CS_i, <>) &= D_{lb\text{-}warp}(<>, S_j) = \infty \\
D_{lb\text{-}warp}(CS_i, S_j) &= D_{lb\text{-}base}(CS_i[1], S_j[1]) + \\
&\quad \min(D_{lb\text{-}warp}(CS_i, S_j[2:-]), D_{lb\text{-}warp}(CS_i[2:-], S_j), D_{lb\text{-}warp}(CS_i[2:-], S_j[2:-])) \\
D_{lb\text{-}base}(CS_i[x], S_j[y]) &= 0 \quad \text{(if } S_j[y] \text{ is included in } CS_i[x]) \\
&= S_j[y] - MAX(CS_i[x]) \quad \text{(if } S_j[y] \text{ is larger than } MAX(CS_i[x])) \\
&= MIN(CS_i[x]) - S_j[y] \quad \text{(if } S_j[y] \text{ is smaller than } MIN(CS_i[x]))
\end{aligned}
$$

As $S_i[x]$ is represented by $CS_i[x]$, the exact distance between $S_i[x]$ and $S_j[y]$ can not be computed. As a lower-bound distance, we use $D_{lb\text{-}base}(CS_i[x],S_j[y])$ that returns the possible minimum distance between $S_i[x]$ and $S_j[y]$. This is shown in Figure 4. Here, $CS_i[x]$ represents the category-id in which the x^{th} element of the sequence S_i is included.

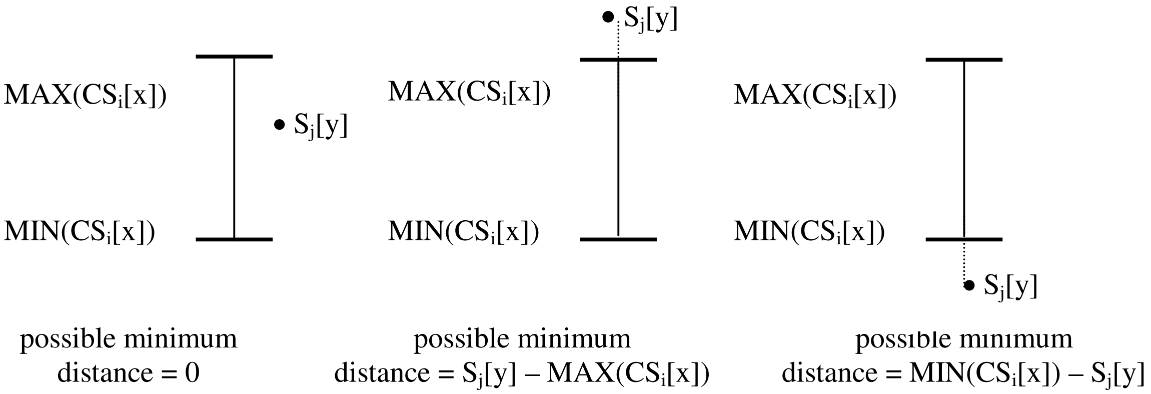
MAX(CSi[x])
MIN(CSi[x])
Sj[y]
possible minimum distance = 0
MAX(CSi[x])
MIN(CSi[x])
Sj[y]
possible minimum distance = Sj[y] – MAX(CSi[x])
MAX(CSi[x])
MIN(CSi[x])
Sj[y]
possible minimum distance = MIN(CSi[x]) – Sj[y]

Figure 4: Possible minimum distance between $CS_i[x]$ and $S_j[y]$

Definition 4 Given any two non-null sequences S_i and S_j, the recurrence relation $\gamma_{lb}(x, y)$ ($x = 1,2,\ldots,Len(S_i)$, $y = 1,2,\ldots,Len(S_j)$) that calculates the lower-bound cumulative distances $D_{lb\text{-}warp}()$ is defined as follows.

$$\gamma_{lb}(0, 0) = 0$$
$$\gamma_{lb}(x, 0) = \gamma_{lb}(0,y) = \infty$$
$$\gamma_{lb}(x, y) = D_{lb\text{-}base}(CS_i[x], S_j[y]) + \min(\gamma_{lb}(x, y-1), \gamma_{lb}(x-1,y), \gamma_{lb}(x-1, y-1))$$

Theorem 2 For any two non-null sequences S_i and S_j, the following inequality holds.

$$D_{lb\text{-}base}(CS_i[x], S_j[y]) \le D_{base}(S_i[x], S_j[y]) \quad (x =1, 2, \ldots,Len(S_i),\ y = 1, 2, \ldots, Len(S_j))$$

Proof The proof is shown in appendix B.

Theorem 3 For any two non-null sequences S_i and S_j, the following inequality holds.

$$D_{lb\text{-}warp}(CS_i, S_j) \le D_{warp}(S_i, S_j)$$

Proof The proof is shown in appendix C.

By Theorem 3, we can guarantee that our similarity search algorithm based on $D_{lb\text{-}warp}()$ does not generate false dismissals. However, the sub-sequences whose time warping distances are larger than ε may be included in the answer-set. They are detected and discarded during post-processing.

Example 4 Using the categorization defined in Example 3, we construct the lower-bound cumulative distance table corresponding to the cumulative distance table shown in Table 2. Y-axis now represents the category-ids of the edges of the categorized suffix tree. The lower-bound time warping distances of Table 4 are computed using $\gamma_{lb}()$.

row 6→	$\mathbb{C}_2$	13	9	9
row 5→	$\mathbb{C}_2$	10	7	7
row 4→	$\mathbb{C}_3$	7	5	5
row 3→	$\mathbb{C}_2$	3	2	2
row 2→	$\mathbb{C}_1$	0	0	*0*
row 1→	$\mathbb{C}_1$	0	0	*0*
		3	4	4

Table 4: Lower-bound cumulative distance table for a query sequence Q = <3,4,4> and the categorized label $<C_1,C_1,C_2,C_3,C_2,C_2>$ corresponding to the label on Y-axis of the cumulative distance table of Table 2

4.3 Sparse Suffix Tree with Categorization

A suffix tree that stores only a subset of suffixes is called *a sparse suffix tree* [13]. Since the size of the suffix tree is linear with respect to the number of leaves, the sparse suffix tree is smaller than an original suffix tree. We call suffixes inserted into a tree *stored-suffixes*, and suffixes not inserted into a tree *non-stored suffixes.* In this work, we insert only suffixes whose first values are different from values of their immediate preceding elements. That is, $S_i[p:-]$ is inserted into a suffix tree only if $S_i[p] \neq S_i[p-1]$. We call a sparse suffix tree constructed from categorized sequences *a categorized sparse suffix tree (CSST).*

Example 5 In Example 3, S_1 and S_2 are transformed to their categorized representations, $CS_1 = <C_1, C_1, C_2, C_3, C_2, C_2>$ and $CS_2 = <C_1, C_2, C_3, C_3>$, respectively. Our categorized sparse suffix tree stores only 7 suffixes ($CS_1[1:-]$, $CS_1[3:-]$, $CS_1[4:-]$, $CS_1[5:-]$, $CS_2[1:-]$, $CS_2[2:-]$, and $CS_2[3]$) from 10 suffixes.

The similarity search algorithm for a categorized suffix tree can also be used on a sparse suffix tree. However, if we use that algorithm without modification, we may miss qualified sub-sequences included in non-stored suffixes. Therefore, we have to find and process non-stored suffixes during tree traversal. Non-stored suffixes can be found easily from the stored suffixes. After finding all the non-stored suffixes, using $D'_{lb\text{-}warp}()$, we compute the lower-bound time warping distance between a query sequence and those non-stored suffixes.

Definition 5 For any two non-null sequences S_i and S_j, if the first N elements of CS_i have same value, then the distance function $D'_{lb\text{-}warp}(CS_i[k:-], S_j)$ (k = 2,3,…,N) that returns the lower-bound distance of $D_{lb\text{-}warp}(CS_i[k:-], S_j)$ is defined as follows:

$$D'_{lb\text{-}warp}(CS_i[k:-], S_j) = D_{lb\text{-}warp}(CS_i, S_j) - (k-1) * D_{lb\text{-}base}(CS_i[1], S_j[1])$$

If we know the value of $D_{lb\text{-}warp}(CS_i, S_j)$, then $D'_{lb\text{-}warp}(CS_i[k:\text{-}], S_j)$ can be computed much faster than the straight computation of $D_{lb\text{-}warp}(CS_i[k:\text{-}], S_j)$.

Theorem 4 For any two non-null sequences S_i and S_j, if the first N elements of CS_i have the same value, then the following inequality holds:

$$D'_{lb\text{-}warp}(CS_i[k:\text{-}], S_j) \le D_{lb\text{-}warp}(CS_i[k:\text{-}], S_j) \le D_{warp}(S_i[k:\text{-}], S_j) \text{ for } k = 2, 3, \ldots, N$$

Proof The proof is shown in appendix D.

4.4 Post-Processing

Since the lower-bound distance functions, $D_{lb\text{-}warp}()$ and $D'_{lb\text{-}warp}()$, are used in our search, the unqualified answers whose time warping distances are larger than a user-given threshold ε may be included in the answer-set. During post-processing, the actual sub-sequences corresponding to answers in the answer-set are retrieved and their time warping distances from a query sequence are calculated. Those sub-sequences whose actual time warping distances are larger than ε are removed from the answer-set.

5. Experiments

To study the performance improvements of our proposed similarity search algorithms, we performed several experiments on 541 stock history data extracted from S&P 500 stock data (http://biz.swcp.com/stocks/) and on the artificial data sequences. The stock data are based on the closing prices of stocks on each day, and the average length of them is 232. The expression for generating the artificial sequences is defined as: $S_i[p] = S_i[p-1] + Z_p$ where Z_p ($p = 1, 2, \ldots$) are independent, identically distributed random variables. The number of and the average length of the artificial sequences vary according to each experiment. We extract 2 query sequences from the stocks whose average prices are below \$30, 5 query sequences from the stocks whose average prices are between \$30 and \$60, and 3 query sequences from the other stocks. The 10 query sequences are extracted from the artificial sequences in a similar manner. The average length of the query sequences from both the stock sequences and the artificial sequences is 20. All experiments except for scalability testing in section 5.3 are performed on both the stock sequences and the artificial sequences. The short notations used in this section are summarized in Table 5.

Symbols	Definitions
SS	Sequential Scanning
CST	Categorized Suffix Tree
CSST	Categorized Sparse Suffix Tree
CSST(N)	Categorized Sparse Suffix Tree having N categories
EL	Categorization Method based on Equal-Length Interval
ME	Categorization Method based on Maximum-Entropy
MC	Categorization Method based on Minimum-Cut

Table 5: Notations used in the experiments

5.1 Index Size and Query Processing Time with Increasing Number of Categories

Table 6 shows the average query processing time (when ε is 30) and size of various suffix trees built from the stock sequences. On the whole, as the number of categories increases, the searches become faster at the cost of bigger index space. However query processing becomes slower when the number of categories exceeds a certain limit, which varies according to the categorization methods. Under the same number of categories, CSST is much smaller than its corresponding CST, and the indexes based on MC are smaller than the indexes based on EL or ME. In Table 6, the boxed, the underlined, and the outlined numbers represent query processing times of the indexes whose sizes are about a half of, same as, and 10 times larger than database size (1,296 Kbytes), respectively. Using similar sized indexes, CSST processes similarity queries faster than CST, and CSST based on ME yields the better performance than CSST based on EL or MC. We have obtained similar conclusions from experiments on artificial sequences.

# categories	Index Size (Kbytes)						Average Query Processing Time (sec) when ε = 30					
	CST			CSST			CST			CSST		
	EL	ME	MC	EL	ME	MC	EL	ME	MC	EL	ME	MC
10	7,513	10,178	5,477	290	618	90	249.38	106.60	632.43	259.60	108.89	593.24
15	8,753	12,166	6,409	434	996	172	189.60	73.98	593.62	173.54	86.17	503.99
20	9,758	14,100	7,516	568	1,425	301	109.93	89.10	235.72	126.25	82.23	240.87
40	13,796	23,408	10,229	1,386	4,711	643	109.15	50.13	138.16	92.34	44.85	139.18
80	21,092	36,321	14,674	3,865	12,195	1661	59.19	44.06	74.50	56.10	39.18	74.89
120	27,808	46,524	19,527	7,100	20,635	3323	50.21	44.59	76.86	42.50	40.68	64.81
160	33,943	53,165	23,641	10,811	27,116	4889	45.33	45.78	61.80	38.31	43.88	52.91
200	39,153	58,795	27,680	14,659	33,743	7080	45.61	48.18	50.46	39.95	48.55	45.36
300	49,440	70,936	35,679	23,678	47,933	12051	44.83	53.05	46.43	40.22	58.91	40.43

Table 6: Average query processing time (when ε = 30) and size of various suffix trees from stock sequences, with an increasing number of categories

5.2 Query Processing Time with Increasing Threshold Values

According to the conclusions from section 5.1, we choose CSST based on ME as our index structure and compare it with sequential scanning with increasing threshold values from 5 to 50. Table 7 and Figure 5 show the experimental results. The same experiments on the artificial sequences have produced similar results. Our proposed technique is up to 4.4 times faster when the index has 10 categories, 7.2 times faster with the index of 20 categories, and 23.2 times faster with the index of 80 categories. Remember that the indexes having 10, 20 and 80 categories require the space about a half of, same as, and 10 times larger than database size, respectively. These results imply that the performance gains of our approach increase as the number of categories increases.

threshold(ε)	Query Processing Time (sec)				Average Number of answers
	SS	CSST (10)	CSST (20)	CSST (80)	
5	408.31	91.97	56.36	17.54	4
10	418.32	98.77	66.25	22.90	1,632
20	426.44	109.04	75.94	31.51	47,446
30	429.11	117.61	82.28	39.19	129,927
40	428.22	125.63	90.60	46.54	214,414
50	428.81	130.69	96.82	53.41	297,598

Table 7: Query processing time and the number of answers with increasing threshold values

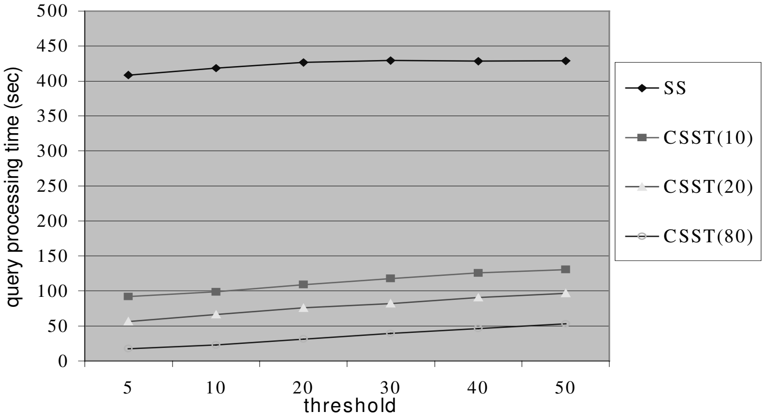

Figure 5: Query processing time with increasing threshold values

5.3 Scalability Testing

To study the scalability of our approach, we compare the query processing time of the CSST approach based on ME with that of sequential scanning, as the average length and the number of the artificial sequences increase. First, we increase the average length of the sequences from 100 to 1,000 while keeping the number of the sequences 200. And, we change the number of sequences from 100 to 10,000 while maintaining the average length of sequences 200. For both experiments, the numbers of categories are chosen to make the size of indexes smaller than the database size and the threshold values are picked to retrieve about 10^{-3} % of the data sub-sequences. As shown in Figure 6 and Figure 7, the performance gain of our approach holds for very long data sequences and a large number of data sequences.

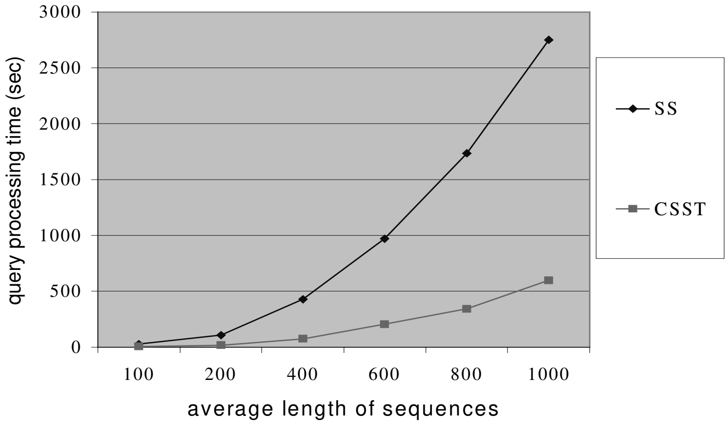

Figure 6. Query processing time with an increasing average length of sequences

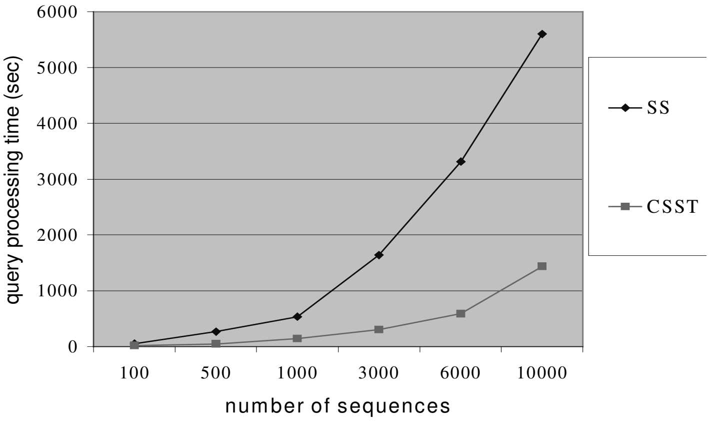

Figure 7. Query processing time with an increasing number of sequences

6. Conclusions

In this paper, we have proposed the indexing technique, which guarantees no false dismissals, for fast similarity retrieval of time-warped sub-sequences. Our method use a suffix tree as an index structure and may be used to answer shape-based queries since the approximate shapes of sub-sequences are maintained in the index space. Experiments on stock and artificial sequences have shown that our approach is about 4 times faster than sequential scanning with a relatively small index space, and the performance gains increase up to 20 times as the size of indexes grows. The contributions of our work are :

- Extending the search algorithm of a suffix tree to similarity matching under time warping
- Applying the concept of categorization and sparse suffix tree to reduce the index size
- Introducing two lower-bound time warping distance functions $D_{lb\text{-}warp}()$ and $D'_{lb\text{-}warp}()$ to query processing

The index space can be reduced further if we know the minimum and maximum lengths of the queries. Using a warping window constraint [11], we can calculate the minimum and maximum lengths of the answers. The suffixes that are shorter than the minimum length of the answers need not be inserted into the suffix trees. For the suffixes that are longer than the maximum, only the prefixes whose lengths are equal to the maximum length need to be inserted into suffix trees.

Our approach can be expanded to multi-dimensional sequences. Under multi-dimensional sequences, the categories are represented as multi-dimensional cells. The same search techniques of a categorized sparse suffix tree can be applied to multi-dimensional cells. We are currently working in this direction for retrieving similar medical image sequences [15].

References

[1] Rakesh Agrawal, Christos Faloutsos, and Arun Swami. Efficient similarity search in sequence databases. In *Proceedings of FODO Conference*, Evanston, IL, USA, October 1993.

[2] Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. Fast subsequence matching in time-series databases. In *Proceedings of the ACM SIGMOD Conference*, May 1994.

[3] Christos Faloutsos and K. Lin. Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proceedings of the ACM SIGMOD Conference*, San Jose, CA, USA, June 1995.

[4] Byoung-Kee Yi, H. V. Jagadish, and Christos Faloutsos. Efficient Retrieval of Similar Time Sequences Under Time Warping. In *International Conference of Data Engineering*, 1998

[5] Dina Q. Goldin and Paris C. Kanellakis. On similarity queries for time-series data: constraint specification and implementation. In *Proceedings of Constraint Programming*, Marseilles, September 1995.

[6] Lawrence Rabiner and Biing-Hwang Juang. Fundamentals of Speech Recognition. Prentice Hall, 1993.

[7] Davood Rafiei and Alberto Mendelzon. Similarity-based queries for time series data. In *Proceedings of the ACM SIGMOD Conference*, Tucson, AZ, May 1997.

[8] Hagit Shatkay and Stanley B. Zdonik. Approximate Queries and Representations for Large Data Sequences, in *Proceedings of Data Engineering Conference*, February 1994.

[9] Rakkesh Agrawal, Giuseppe Psaila, Edward L. Wimmers and Mohamed Zaït. Querying Shapes of Histories, in *Proceedings of the 21st VLDB Conference*, Zürich, Switzerland, 1995

[10] Tolga Bozkaya, Nasser Yazdani, and Meral Özsoyoğlu. Matching and Indexing Sequences of Different Lengths, in *Proceedings of CIKM*, Las Vegas, NV, 1997

[11] Donald J. Berndt and James Clifford. Finding Patterns in Time Series, Advances in Knowledge Discovery and Data Mining, AAAI/MIT Press, 1996

[12] Graham A. Stephen. String Searching Algorithms. World Scientific Publishing Co., 1994

[13] Juha Kärkkäinen and Esko Ukkonen. Sparse Suffix Trees, in *Proceedings of COCOON*, HongKong, 1996

[14] C. E. Shannon and W. Weaver. The Mathematical Theory of Communication. Urbana, Ill.: University of Illinois Press, 1964

[15] Wesley W. Chu, Alfonso F. Cardenas, and Ricky K. Taira. KMeD: a Knowledge-based Multimedia Medical Distributed Database System, *Information Systems*, Vol.20, No.2, Premagon-Press/Elsevier Science, 1995

Appendix A

Theorem 1 If all columns of the last row of the cumulative distance table have values greater than a user-given threshold ε, adding more rows to this cumulative distance table does not yield any new answers.

Proof

Let us assume that the cumulative distance table is being constructed for two non-null sequences S_i and S_j, and S_i is located in Y-axis and S_j in X-axis. Then, Theorem1 can be re-written formally as:

$$\text{If } \gamma(m, n) > \varepsilon \text{ for any } m\ (m=1,2,\ldots,Len(S_i-1)) \text{ and for all } n\ (n=1,2,\ldots,Len(S_j)), \qquad \text{(a1)}$$
$$\text{then } \gamma(m+1, n) > \varepsilon.$$

We prove Theorem 1 by induction.

We assume that (a1) is TRUE. (a2)

By the definition of $\gamma()$, $\gamma(m+1, 1)$ is represented as:

$$\gamma(m+1, 1) = D_{base}(S_i[m+1], S_j[1]) + \gamma(m, 1)$$

Since

$D_{base}(S_i[m+1], S_j[1]) \geq 0$ and (by the definition of $D_{base}()$),

$\gamma(m, 1) > \varepsilon$ (by (a2)),

we have $\gamma(m+1, 1) > \varepsilon$ (a3)

We assume that $\gamma(m+1, k) > \varepsilon$ for any k (k=1,2,…,Len(S_j) –1). (a4)

By the definition of $\gamma()$, $\gamma(m+1, k+1)$ is represented as:

$$\gamma(m+1, k+1) = D_{base}(S_i[m+1], S_j[k+1]) + \min(\gamma(m, k+1), \gamma(m+1, k), \gamma(m, k))$$

Since

$D_{base}(S_i[m+1], S_j[k+1]) \geq 0$ (by the definition of $D_{base}()$),

$\gamma(m, k+1) > \varepsilon$ (by (a2)),

$\gamma(m+1, k) > \varepsilon$, and (by (a4)),

$\gamma(m, k) > \varepsilon$ (by (a2)),

we have

$$\gamma(m+1, k+1) > \varepsilon \quad \text{(a5)}$$

Based on (a2), (a3), (a4) and (a5), Theorem 1 is TRUE.

Appendix B

Theorem 2 For any two non-null sequences, S_i and S_j, the following inequality holds.

$$D_{lb\text{-}base}(CS_i[x], S_j[y]) \leq D_{base}(S_i[x], S_j[y]) \quad (x = 1,2,\ldots,Len(S_i),\ y=1,2,\ldots,Len(S_j))$$

Proof

By definition of $CS_i[x]$, we know that $MIN(CS_i[x]) \leq S_i[x] \leq MAX(CS_i[x])$.

There are three possible $D_{lb\text{-}base}()$ expressions according to the values of $CS_i[x]$ and $S_j[y]$.

Case 1 : If $MIN(CS_i[x]) \leq S_j[y] \leq MAX(CS_i[x])$, then

$$D_{lb\text{-}base}(CS_i[x], S_j[y]) = 0 \leq | S_i[x] - S_j[y] | = D_{base}(S_i[x], S_j[y])$$

Case 2 : If $MAX(CS_i[x]) < S_j[y]$, then

$$D_{lb\text{-}base}(CS_i[x], S_j[y]) = S_j[y] - MAX(CS_i[x]) \leq | S_j[y] - S_i[x] | = D_{base}(S_i[x], S_j[y])$$

Case 3 : If $MIN(CS_i[x]) > S_j[y]$, then

$$D_{lb\text{-}base}(CS_i[x], S_j[y]) = MIN(CS_i[x]) - S_j[y] \leq | S_i[x] - S_j[y] | = D_{base}(S_i[x], S_j[y])$$

For all possible values of $CS_i[x]$ and $S_j[y]$, Theorem 2 is TRUE.

Appendix C. Theorem 3

Theorem 3 For any two non-null sequences S_i and S_j, the following inequality holds.

$$D_{lb\text{-}warp}(CS_i, S_j) \le D_{warp}(S_i, S_j)$$

Proof

The inequality of Theorem 3 can be re-written using the recurrence relation for cumulative distance table.

$$\gamma_{lb}(M, N) \le \gamma(M, N)\,(M = Len(S_i),\ N = Len(S_j))$$

As the formal proof of Theorem 3 is long, we just show the sketch of the proof.

By Theorem 2 and the definitions $\gamma_{lb}()$ and $\gamma()$,

$$\gamma_{lb}(1, 1) \le \gamma(1, 1). \tag{c1}$$

Using induction process from (c1), we get

$$\gamma_{lb}(1, n) \le \gamma(1, n) \text{ for all } n \ (n=1,2, \ldots, Len(S_j)). \tag{c2}$$

Using induction process from (c2), we get

$$\gamma_{lb}(m, n) \le \gamma(m, n) \text{ for all } m \text{ and } n \ (m=1,2,\ldots,Len(S_i),\ n=1,2, \ldots, Len(S_i)) \tag{c3}$$

By (c3), Theorem 3 is TRUE.

Appendix D. Theorem 4

Theorem 4 For any two not-null sequences S_i and S_j, if the first N elements of CS_i have the same value, then the following inequality holds.

$$D'_{lb\text{-}warp}(CS_i[k:-], S_j) \le D_{lb\text{-}warp}(CS_i[k:-], S_j) \le D_{warp}(S_i[k:-], S_j) \text{ for } k = 2, 3, \ldots, N$$

Proof

Let us assume that the first N elements of CS_i have the same value.

By the definition of $D_{lb\text{-}warp}()$, we know that

$$D_{lb\text{-}warp}(CS_i[2:-], S_j) \ge D_{lb\text{-}warp}(CS_i, S_j) - D_{lb\text{-}base}(CS_i[1], S_j[1]). \tag{d1}$$

Let us assume that for any m ($m = 2, 3, \ldots, N-1$),

$$D_{lb\text{-}warp}(CS_i[m:-], S_j) \ge D_{lb\text{-}warp}(CS_i, S_j) - (m-1) * D_{lb\text{-}base}(CS_i[1], S_j[1]), \tag{d2}$$

Then, we have

$$\begin{aligned} D_{lb\text{-}warp}(CS_i[m+1:-], S_j) &\ge D_{lb\text{-}warp}(CS_i[m:-], S_j) - D_{lb\text{-}base}(CS_i[m], S_j[1]) = \\ &\quad D_{lb\text{-}warp}(CS_i[m:-], S_j) - D_{lb\text{-}base}(CS_i[1], S_j[1]) \\ &\ge D_{lb\text{-}warp}(CS_i, S_j) - (m-1) * D_{lb\text{-}base}(CS_i[1], S_j[1]) - D_{lb\text{-}base}(CS_i[1], S_j[1]) = \\ &\quad D_{lb\text{-}warp}(CS_i, S_j) - m * D_{lb\text{-}base}(CS_i[1], S_j[1]) \end{aligned}$$

That is, $D_{lb\text{-}warp}(CS_i[m+1:-], S_j) \ge D_{lb\text{-}warp}(CS_i, S_j) - m * D_{lb\text{-}base}(CS_i[1], S_j[1])$ (d3)

By (d1), (d2) and (d3), we know that, for all k ($k = 2, 3, \ldots, N$),

$$D_{lb\text{-}warp}(CS_i, S_j) - (k-1) * D_{lb\text{-}base}(CS_i[1], S_j[1]) \le D_{lb\text{-}warp}(CS_i[k:-], S_j)$$

That is, for all k ($k = 2, 3, \ldots, N$),

$$D'_{lb\text{-}warp}(CS_i[k:-], S_j) \le D_{lb\text{-}warp}(CS_i[k:-], S_j)\ (2 \le k \le N) \tag{d4}$$

By (d4) and Theorem 3, Theorem 4 is TRUE.