Efficient Searches for Similar Subsequences of Different Lengths in Sequence Databases *

Sanghyun Park and Wesley W. Chu
University of California, Los Angeles
shpark@cs.ucla.edu, wwc@cs.ucla.edu

Jeehee Yoon
Hallym University, Korea
jhyoon@sun.hallym.ac.kr

Chihcheng Hsu
Santa Teresa Lab., IBM
ccheng@us.ibm.com

Abstract

We propose an indexing technique for fast retrieval of similar subsequences using time warping distances. A time warping distance is a more suitable similarity measure than the Euclidean distance in many applications, where sequences may be of different lengths or different sampling rates. Our indexing technique uses a disk-based suffix tree as an index structure and employs lower-bound distance functions to filter out dissimilar subsequences without false dismissals. To make the index structure compact and thus accelerate the query processing, we convert sequences of continuous values to sequences of discrete values via a categorization method and store only a subset of suffixes whose first values are different from their preceding values. The experimental results reveal that our proposed technique can be a few orders of magnitude faster than sequential scanning.

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 lesions have similar evolution characteristics are a few examples of similarity queries. Although sequential scanning can be used to answer these queries, it may require an enormous processing time over large sequence databases. Recently, several indexing techniques [1,5,10,22] have been proposed to speed up the processing of similarity queries.

Most of the previous techniques [1,10,22] for similarity searches use the Euclidean distance metric as a similarity measure. However, in many applications, the sampling rates and the lengths of sequences may be different, making it difficult or impossible to use the Euclidean distance as a similarity measure. In the area of speech recognition [15], this problem has been approached using a similarity measure, called a time warping distance [3,15], which allows sequences to be stretched or compressed along the time axis. Under time warping, any element of a sequence can be matched to one or more neighboring elements of another sequence. As an example [16], let us consider two sequences, S_1 = <20,20,21,21,20,20,23,23> and S_2 = <20,21,20,23> where the sequence S_1 is the closing price of a stock taken every day and S_2 is the closing price of another stock taken every other day. S_1 and S_2 cannot be compared directly because the sequence S_1 is longer than S_2. The Euclidean distance between S_2 and any subsequence of length four of S_1 is greater than 1.41. However, if we duplicate every element of S_2 using time warping, we find that the two sequences are identical.

In the matching of similar sequences, it is important to prevent the occurrence of *false dismissals* [1]. A false dismissal occurs when a sequence similar to a query sequence is not included in the answer set. Indexing techniques that assume the *triangular inequality* may produce false dismissals when the distance function not satisfying the triangular inequality is used as a similarity measure [22]. Unfortunately, a time warping distance does not satisfy the triangular inequality, which can be simply proved by a counter example [22]. This property makes spatial access methods based on the triangular inequality unsuitable for similarity searches with a time warping distance.

In the area of string matching, a suffix tree [17] has been extensively used as an index structure to find the substrings that are exactly matched to the given query string. A suffix tree may be a good candidate for an index structure with a time warping distance because it does not assume any geometry or any underlying distance functions. However, for a suffix tree to be used as an index structure for similarity searches, the following problems have to be addressed: 1) A suffix tree is designed to find the exactly matched substrings. Its exact search algorithm needs to be extended to find similarly matched subsequences. 2) In general, a suffix tree is built

* This work is supported in part by grants from the NSF (IRI--9619345) and the NIH (CA51198-07)

from sequences whose elements take the values from finite alphabets. However, sequences we consider in this paper are comprised of elements of continuous real values. A systematic method to convert continuous element values to discrete values is required.

In this paper we propose a new indexing technique for the fast retrieval of similar subsequences of different lengths or different sampling rates. Our technique uses a time warping distance as a similarity measure and a disk-based suffix tree as an index structure. To make the index structure compact, we convert sequences of continuous values to sequences of discrete values via a categorization method and store only a subset of suffixes whose first values are different from their immediately preceding values. When the query sequence, Q, is given, a suffix tree is traversed and time warping distances between Q and subsequences contained in a suffix tree are computed. Because subsequences contained in a suffix tree are of discrete values, their exact distances from Q cannot be obtained. Instead, lower-bound distance functions are employed to estimate the exact distance; so our proposed technique guarantees no false dismissals.

This paper is organized as follows. In Section 2 we provide a brief overview of the related work on sequence matching problems. In Section 3, we give the definition and the property of a time warping distance. Section 4 introduces the construction method and the similarity search algorithm of a disk-based suffix tree. We apply the ideas of a categorization and a sparse suffix tree in Section 5 and Section 6, respectively. Experimental results are given in Section 7.

2. Related work

Several approaches for fast retrieval of similar sequences have recently been proposed. In [1], sequences are converted into the frequency domain by a Discrete Fourier Transform and are subsequently mapped into multi-dimensional points that are managed by an R*-tree; this technique was extended to locate similar subsequences [10]. Since the approaches of [1,10] use the Euclidean distance metric as a similarity measure, sequences of different lengths or different sampling rates cannot be matched.

Sequence matching that allows transformations is proposed in [11,16]. In [11], sequences are grouped into equivalent classes according to their normal forms. Though normal forms are invariant to shape-based transformations such as scaling and shifting, they do not handle the compressions or the stretches of element values along the time axis. The authors of [16] propose a class of transformations that can be used in a query language to express similarity with an R-tree index. They handle moving average and global time scaling, but not time warping.

The access methods of [5,14,21,22] permit the matching of sequences of different lengths. [5] presents a modified version of an edit distance, considering two sequences matching if a majority of elements match. This technique is extended to the matching of multi-dimensional sequences in [21]. In [22], a time warping distance is used as a similarity measure with a two-step filtering process: a FastMap index filter proceeded by a lower-bound distance filter. The underlying index structures of [5,21,22] are based on the triangular inequality. The authors of [14] introduce an aligned subsequence matching with a time warping distance. Whereas their approaches are useful for long data sequences, subsequences not starting or ending at segment boundaries cannot be found.

Similarity matching based on shapes of sequences is proposed in [2,19]. [2] demonstrates a shape definition language (SDL) and provides an index structure for speeding up the execution of SDL queries. In [19], the authors introduce the notion of generalized approximate queries that specify the general shapes of data histories. Whereas both approaches may handle the variations of element values on the time axis, 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 suffix tree for solving the sequence alignment problem, and [20] 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 continuous numeric values, the approaches of [4,20] center on the sequences of characters. Furthermore, the algorithm of [20] uses a main-memory based suffix tree, making it infeasible for a large sequence set.

3. Time warping distance

In general, finding a similarity measure for sequences is not easy because sequences that are qualitatively the same may be quantitatively different. First, the sequences may be of different lengths, making it difficult or impossible to embed the sequences in a metric space and use the Euclidean distance to determine similarity. Second, the sampling rates of sequences may be different: one sequence may be sampled every minute while another sequence is sampled every other minute. Such differences in rates make similarity measures such as cross-correlation unusable.

In this paper, we use a time warping (TW) similarity measure [3,15] that allows sequences to be stretched or compressed along the time axis. TW is a generalization of classical algorithms for comparing discrete sequences to sequences of continuous values, and is used extensively in

matching of voice, audio and medical signals (electrocardiograms). To find the minimum difference between two sequences, TW maps each element of a sequence to one or more neighboring elements of another sequence. Let us now give the formal definition [15] of the time warping distance.

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

$$D_{tw}(S_i, S_j) = D_{base}(S_i[1], S_j[1]) + \min \begin{cases} D_{tw}(S_i, S_j[2:-]) \\ D_{tw}(S_i[2:-], S_j) \\ D_{tw}(S_i[2:-], S_j[2:-]) \end{cases}$$

$$D_{base}(a, b) = | a - b | \ \blacksquare$$

In above definition, $S_i[p]$ represents the p^{th} element of S_i and $S_i[p:q]$ denotes the subsequence of S_i including elements in positions p through q. We use the notation $S_i[p:-]$ for the suffix of S_i starting from the p^{th} element. That is, $S_i[p:-]$ is identical to $S_i[p:|S_i|]$ where $|S_i|$ is the length of S_i. $D_{base}()$ on two numeric values can be any distance function, but we assume that it is defined as the city-block distance. $D_{tw}(S_i,S_j)$ can be calculated efficiently using a dynamic programming technique [3] based on the recurrence relation $\gamma_{tw}(x, y)$.

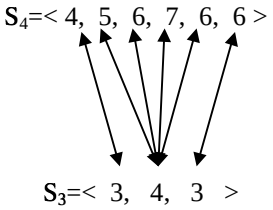
Definition 2. Given any two non-null sequences, S_i and S_j, the recurrence relation $\gamma_{tw}(x, y)$ ($x=1,2,\ldots,|S_i|$, $y=1,2,\ldots,|S_j|$) that builds the cumulative time warping distance table for S_i and S_j is defined as follows:

$$\gamma_{tw}(x, y) = D_{base}(S_i[x], S_j[y]) + \min \begin{cases} \gamma_{tw}(x, y-1) \\ \gamma_{tw}(x-1, y) \\ \gamma_{tw}(x-1, y-1) \end{cases}$$

$\blacksquare$

row6	6	16	11	**12**
row5	6	13	9	10
row4	7	10	7	8
row3	6	6	4	5
row2	5	32	_	3
row1	4	1	1	2
	S_4 / S_3	3	4	3
		col1	*col2*	*col3*

(a) Cumulative distance table



(b) Mapping of elements

Figure 1. Time warping distance for S_3 = <3,4,3> and S_4 = <4,5,6,7,6,6>

The dynamic programming algorithm [3] fills in the table of cumulative distances as the computation proceeds. This computation has complexity $O(|S_i||S_j|)$. The final cumulative distance, $\gamma_{tw}(|S_i|,|S_j|)$, is the minimum distance between S_i and S_j, and the matching of elements can be traced backward in the table by choosing the previous cells with the lowest cumulative distance. Figure 1 shows the cumulative distance table for two sequences, S_3 = <3,4,3> and S_4 = <4,5,6,7,6,6> and the mapping of elements that generates the minimum distance.

By reading the last column of each row of the cumulative distance table, we get the distance between S_i and any prefix of S_j. That is, the distance between S_i and $S_j[1:q]$ ($q=1,2,\ldots,|S_j|$) is obtained from the last column of the q^{th} row. In the above example, $D_{tw}(S_3, S_4[1:4])$ is 8, as seen in the last column of the row 4. Thus, determination as to whether the time warping distance of two sequences is greater than a distance-threshold ε does not require building the entire cumulative distance table, as proven by the following theorem.

Theorem 1. If all columns of the last row of the cumulative distance table have values greater than a distance-threshold ε, adding more rows on this table does not yield the new values less than or equal to ε.
Proof. The proof is given in [13]. $\blacksquare$

Let us look at the table shown in Figure 1. If ε is 3, after inspecting the row 3, we can determine that the distance between S_3 and S_4 is greater than ε because all columns of the row 3 have values greater than 3. Therefore, we do not have to fill the remaining three rows. In subsequent sections, we use Theorem 1 to reduce the search space of an index structure.

4. Similarity search using a suffix tree

In this section, we propose to use a suffix tree (ST) as an index structure for similarity searches with a time warping distance. Before describing the methods for constructing and searching a suffix tree, we present the definition and the internal structure of a suffix tree.

A trie is an indexing structure used for indexing sets of keywords of varying sizes. A suffix trie [17] 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 [17]. A suffix tree is generalized [4,20] to allow multiple sequences to be stored in the same tree. Each suffix of a sequence is represented by a leaf node. Precisely, the suffix $S_t[p:-]$ is expressed by a leaf node labeled with (t,p). The edges are labeled with subsequences such that the concatenation of the edge labels on the path from the root to the leaf (t,p) becomes $S_t[p:-]$. 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. Figure 2 shows the suffix tree constructed from two sequences, S_5 = <4,5,6,7,6,6> and S_6 = <4,6,7,8>, where '$' is used as an end marker of a suffix.

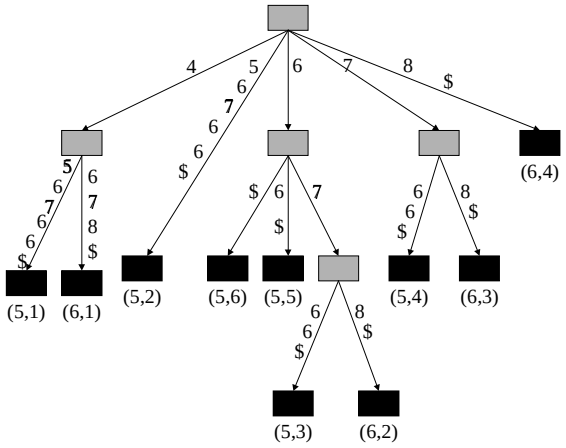


Figure 2. Suffix tree from S_5=<4,5,6,7,6,6> and S_6=<4,6,7,8>

4.1. Index construction

A suffix tree for multiple sequences can be constructed by adding a special sequence separator symbol to the alphabet. The sequences to be included in the tree are concatenated, separated from each other by this separator symbol. Then, the ordinary suffix tree algorithm is applied to the concatenated sequence. The suffix tree created using this process has to be kept in main memory during construction. Therefore, this approach is not realistic to a large sequence set.

To remedy the problem, we use an incremental disk-based suffix tree construction method proposed in [4]. Two suffix trees, representing two disjoint sets of sequences, are merged to produce a single suffix tree by pre-order traversal of both suffix trees and combining the paths corresponding to common subsequences. A suffix tree for a large set of sequences can be constructed by performing a series of binary merges of suffix trees of increasing size. The merge operation of two suffix trees can support disk-based representations in limited main memory.

Two suffix trees for S_i and S_j are merged with complexity $O(|S_i|+|S_j|)$, hence the suffix tree for M sequences is constructed with complexity $O(M\bar{L})$ where $\bar{L}$ is the average length of M sequences. The total number of nodes in a suffix tree is constrained due to two factors: 1) there are $O(M\bar{L})$ leaf nodes and 2) 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 $M\bar{L}$ [17].

4.2. Search algorithm: SimSearch-ST

A suffix tree (ST) is a useful index structure to locate subsequences that are exactly matched to a query sequence Q. To find exactly matched subsequences, Q is traversed from the root of the tree and traversal is terminated when the end of Q is reached or a node is reached beyond which further traversal is not possible. Exact searches are performed in O(|Q|). Even though the exact matching algorithm of a suffix tree is simple and fast, it cannot be directly applied to the problem we are going to solve in this paper.

Problem Definition: Given M sequences $S_1,S_2,\ldots,S_M$ of arbitrary lengths, a query sequence Q and a user-given distance-threshold ε, find the subsequences $S_i[p:q]$ (i = 1,2,…,M) whose time warping distances from Q are less than or equal to ε. ■

Our proposed similarity search algorithm SimSearch-ST is given in Algorithm 1. The search starts from the root of a suffix tree and continues the depth-first traversal until all subsequences whose time warping distances are less than or equal to ε are found.

```
Input  : Root Node R, Q, ε
Output : answerSet

cumDistTable ← NULL;
answerSet ← Filter-ST (R, Q, ε, cumDistTable);

return answerSet;
```

Algorithm 1. SimSearch-ST

The actual filtering process is executed in Filter-ST shown in Algorithm 2. When Filter-ST visits a node N, it inspects each child node CN_i to find a new answer and to determine whether further depth traversal is needed or not. For simpler explanation, we assume that the edge between two nodes, N and CN_i, is labeled with a single value.

To find a new answer, Filter-ST builds a cumulative distance table for Q and label(N, CN_i). If N is a root node, the table is built from the bottom. Otherwise, the table is constructed by augmenting a new row on the table T that has been accumulated from the root to N. The function AddRow(T, Q, label(N, CN_i), D_{tw}()) builds a new cumulative distance table, using the distance function D_{tw}(), by augmenting a new row corresponding to label(N, CN_i) on T. Suppose that the r^{th} row is the newly added row. If the last column of the r^{th} row has the value less

than or equal to ε, label(GetRoot(CN_i), CN_i)) is inserted into the answer set.

To determine if further depth traversal is needed, Filter-ST uses Theorem 1. If at least one column of the r^{th} row has a value not greater than ε, the search continues down the tree to find more answers. Otherwise, the search moves to the next child of N.

```
Input  : Node N, Q, ε, Cumulative Distance Table T
Output : answerSet

answerSet ← {};
CN ← GetChildren(N)

for i ← 1 to |CN| do {
    CT_i ← AddRow(T, Q, label(N,CN_i), D_tw());
    Let dist be the last column value of the new row;
    Let mDist be the minimum column value of the new row;
    if dist ≤ ε, insert label(GetRoot(CN_i),N_i) into answerSet;
    if mDist ≤ ε,
        answerSet ← answerSet ∪ Filter-ST(CN_i, Q, ε, CT_i);
}

return answerSet;
```

Algorithm 2. Filter-ST

4.3. Algorithm analysis

Before analyzing the complexity of SimSearch-ST, let us examine the complexity of sequential scanning. Sequential scanning reads each sequence and builds as many cumulative distance tables as the number of suffixes contained in the sequence. The complexity of building a cumulative distance table for the query sequence Q and the suffix of length L is O(L|Q|). For M sequences whose average length is $\bar{L}$, there are M $\bar{L}$ suffixes and their average length is ($\bar{L}$+1)/2. Therefore, the complexity of sequential scanning is $O(M\bar{L}^2|Q|)$.

SimSearch-ST is computationally less expensive than sequential scanning due to branch-pruning (based on Theorem 1) and sharing cumulative distance tables for all suffixes that have common prefixes. Thus, the complexity of SimSearch-ST is $O(\frac{M\bar{L}^2|Q|}{R_d R_p})$, where R_d (≥1) is the reduction factor due to sharing the cumulative distance tables, and R_p (≥1) is the reduction factor gained from the branch-pruning. R_d grows as the length and the number of common prefixes of the suffixes contained in a suffix tree increase. Given k suffixes, $\alpha_1,\alpha_2,\ldots,\alpha_k$, whose first t elements are the same, the construction of k cumulative distance tables requires the computation of $|Q||\alpha_1| + |Q||\alpha_2| + \ldots + |Q||\alpha_k|$ cells. However, it is reduced to $t|Q| + |Q|(|\alpha_1|-t) + |Q|(|\alpha_2|-t) + \ldots + |Q|(|\alpha_k|-t)$ if the cumulative distance table for Q and the common prefix of length t is shared by k suffixes. In this case, R_d can be expressed as $R_d = (|\alpha_1|+|\alpha_2|+\ldots+|\alpha_k|) / ((|\alpha_1|+|\alpha_2|+\ldots+|\alpha_k|) - t(k-1))$.

While R_d is determined by the distribution of element values, R_p is decided by the number of answers required by a user. That is, R_p increases as the distance-threshold ε decreases. If ε is so small that just one or two subsequences may be answers, only the topmost part of a suffix tree may be visited during the query processing. In another extreme case where ε is large enough for all subsequences to be answers, all nodes of a suffix tree need to be visited, thus making $R_p = 1$. In the worst case where there is no common subsequence and the branch-pruning cannot help, both values of R_d and R_p are 1, and therefore the complexity of SimSearch-ST becomes the same as that of sequential scanning.

5. Similarity search using categorization

In this section we introduce the concept of categorization to decrease the number of values that elements can take and thus increase the length and the number of common subsequences. As explained in the previous section, if the length and the number of common subsequences increase, the index size and the query processing time are reduced. To get the categorized representations of element values, we first generate the set of categories and determine their ranges. Then, we convert every element value to the symbol of the corresponding category. For example, given two categories C_1 = [0.1,3.9] and C_2 = [4.0,10.0], S_7 = <5.27, 2.56, 3.85> is transformed to CS_7 = <C_2, C_1, C_1> where CS_7 denotes the converted sequence of S_7. Thus, sequences of continuous values are converted to sequences of discrete symbols.

5.1. Categorization method

In this work, the following two categorization methods have been chosen and experimented for their simple implementations, albeit other categorization approaches like the type abstraction hierarchy (TAH) [6] and the k-means algorithm may also be used to categorize element values.

The first method is an equal-length categorization. As the name implies, all the categories have equal interval length (MAX-MIN) / c. Here, MIN is the smallest element value found in the set of sequences, MAX is the largest element value found in the set of sequences, and c is the number of categories given as the input parameter to the categorization algorithm. This categorization approach is simple and fast, but loses information about value and frequency distributions of the sequences.

The second method is a maximum-entropy categorization. The entropy [18] of a categorization is defined as $H(C) = -\sum_{i=1}^{c} P(C_i) \log P(C_i)$ where $P(C_i)$ is the probability that an element is included in the i^{th} category. To minimize the loss of information about the sequences, this categorization method decides the category boundaries that generate the maximum entropy value. The boundaries can be determined easily by making all categories include the same number of elements ($P(C_1) = P(C_2) = \ldots = P(C_c)$).

It is not easy to determine the number of categories: too many categories do not help much to increase the number of common subsequences, but likewise, too few categories do not help much to reduce the query processing time because of the decreased filtering rate of the index. A simple strategy is to do many experiments on the set of sequences and determine the best number of categories using the cost function $W_t C_t + W_s C_s$. Here, C_t and C_s are costs for processing the query and storing the index, respectively, and W_t and W_s are their relative weights. The determination of these weights is application-dependent.

5.2. Index construction

After converting element values into discrete symbols, we build a suffix tree from the set of converted sequences. We denote the resultant tree ST_C. ST_C is constructed using the same construction algorithm used for an ordinary suffix tree.

5.3. A modified distance function: $D_{tw\text{-}lb}()$

Whereas the edges of a suffix tree are labeled with numeric values, the edges of ST_C are labeled with discrete symbols. As a result, the exact time warping distance between a query sequence of numeric values and any subsequence contained in ST_C cannot be computed. Therefore, we introduce the new distance function $D_{tw\text{-}lb}()$.

Definition 3. Given two non-null sequences, S_i and S_j, the distance function $D_{tw\text{-}lb}(S_i, CS_j)$ that returns the lower-bound distance of $D_{tw}(S_i, S_j)$ is defined as follows:

$$D_{tw\text{-}lb}(S_i, CS_j) = D_{base\text{-}lb}(S_i[1], CS_j[1]) + \min \begin{cases} D_{tw}(S_i, CS_j[2:-]) \\ D_{tw}(S_i[2:-], CS_j) \\ D_{tw}(S_i[2:-], CS_j[2:-]) \end{cases}$$

$$D_{base\text{-}lb}(a, B) = \begin{cases} 0 & (\text{if } B.lb \le a \le B.ub) \\ a - B.ub & (\text{if } a > B.ub) \\ B.lb - a & (\text{if } a < B.lb) \end{cases}$$

where 'a' is the numeric value corresponding to $S_i[1]$ and 'B' is the category symbol corresponding to $CS_j[1]$. ■

In the definition of $D_{base\text{-}lb}()$, B.lb and B.ub are the minimum and the maximum element values, respectively, found in the category B. As shown in Figure 3, $D_{base\text{-}lb}(a, B)$ returns the possible minimum distance between a and B.

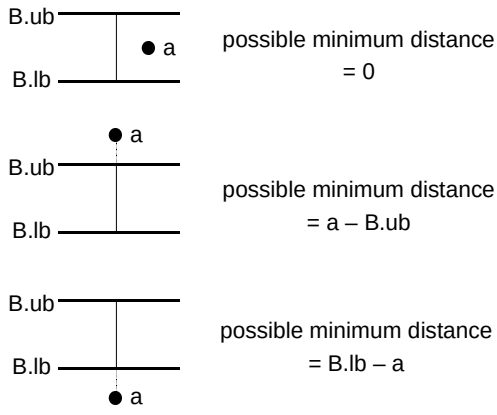


Figure 3. Minimum distance between a and B

To prevent false dismissals, the distance returned from $D_{tw\text{-}lb}(S_i, CS_j)$ should always be less than or equal to the distance computed by $D_{tw}(S_i, S_j)$. Theorem 2 states this fact.

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

$$D_{tw\text{-}lb}(S_i, CS_j) \le D_{tw}(S_i, S_j)$$

Proof. The proof is given in [13]. ■

5.4. Search algorithm: SimSearch-ST_c

```
Input  : Root Node R, Q, ε
Output : answerSet

cumDistTable ← NULL;
candidateSet ← answerSet ← {};

candidateSet ← Filter-ST_C (R, Q, ε, cumDistTable);
answerSet ← PostProcess(candidateSet, Q, ε)

return answerSet;
```

Algorithm 3. SimSearch-ST_c

The algorithm SimSearch-ST needs to be modified to reflect the categorized representation of element values. Our proposed search algorithm SimSearch-ST_C is shown in Algorithm 3. Note that element values of a query sequence are not converted to discrete symbols.

To find the candidate subsequences whose lower-

bound distances to the query sequence Q are within ε, Filter-ST_c is called recursively. Filter-ST_c is the same as Filter-ST except that the former uses $D_{tw\text{-}lb}()$ to build the cumulative distance table while the latter uses $D_{tw}()$. Since the lower-bound distance is used for filtering, the subsequences whose exact time warping distances are larger than ε may be included in the candidate answer set. These subsequences are called *false alarms*. For each answer contained in the candidate answer set, the algorithm PostProcess retrieves the actual subsequences and computes their exact time warping distances. Finally, the subsequences whose actual time warping distances are not larger than ε are returned as answers. Algorithms Filter-ST_c and PostProcess are omitted due to space limitations.

5.5. Algorithm analysis

The complexity of SimSearch-ST_c is represented as $O(\frac{M\bar{L}^2|Q|}{R_d R_p} + n\bar{L}|Q|)$ where n is the number of subsequences requiring the post-processing. Hence, the left expression represents the time for filtering and the right expression represents the time for post-processing. Compared to SimSearch-ST, SimSearch-ST_c has performance improvements due to a larger value of R_d, despite the extra time for post-processing

6. Similarity search using a sparse suffix tree

A suffix tree that stores only a subset of suffixes is called a sparse suffix tree [12]. Since the size of a suffix tree is linear in the number of leaves, a sparse suffix tree is smaller than an original suffix tree. Suffixes inserted into a tree are called *stored suffixes*, and suffixes not inserted into a tree are called *non-stored suffixes*. The reduction of the index size by storing only a subset of suffixes is measured by *the compaction ratio r* ($0 \le r < 1$) that is defined as r = (the number of non-stored suffixes) / (the number of suffixes). In this section, we propose to use a sparse suffix tree to further reduce the index size and accelerate the query processing.

6.1. Index construction

Similar to ST_C, a sparse suffix tree is built from the set of categorized sequences. However, unlike ST_C, only suffixes whose first values are different from their immediately preceding values are stored in a sparse suffix tree. That is, the suffix $CS_j[p:-]$ is stored only if $CS_j[p] \neq CS_j[p-1]$. For example, for $CS_8 = \langle C_1,C_1,C_1,C_3,C_2,C_2\rangle$, only the three suffixes ($CS_8[1:-]$, $CS_8[4:-]$, and $CS_8[5:-]$) are stored in a sparse suffix tree. We denote the resultant tree SST_C.

6.2. A modified distance function: $D_{tw\text{-}lb2}()$

Suppose that we have the cumulative distance table for S_i and CS_j where S_i and CS_j are located along the x-axis and the y-axis, respectively. While we can get the distance between S_i and any *prefix* of CS_j by reading the last column of each row, there is no direct way to compute the distance between S_i and any *suffix* of CS_j except by building a new table. However, if the first N elements of CS_j have the same value, we can obtain the lower-bound distance of $D_{tw\text{-}lb}(S_i, CS_j[p:-])$ (p=2,3,…,N) using a new distance function, $D_{tw\text{-}lb2}(S_i, CS_j[p:-])$.

Definition 4. For any two non-null sequences, S_i and CS_j, if the first N elements of CS_j have the same value, then the distance function $D_{tw\text{-}lb2}(S_i, CS_j[p:-])$ (p=2,3,…,N) that returns the lower-bound distance of $D_{tw\text{-}lb}(S_i, CS_j[p:-])$ is defined as follows:

$$D_{tw\text{-}lb2}(S_i,CS_j[p:-]) = D_{tw\text{-}lb}(S_i,CS_j) - (p-1) * D_{base\text{-}lb}(S_i[1],CS_j[1])$$

■

If we know the value of $D_{tw\text{-}lb}(S_i,CS_j)$, then $D_{tw\text{-}lb2}(S_i, CS_j[p:-])$ can be computed with complexity O(1). The distance returned from $D_{tw\text{-}lb2}(S_i,CS_j[p:-])$ is always less than or equal to $D_{tw\text{-}lb}(S_i, CS_j[p:-])$. The following theorem states this fact.

Theorem 3. For any two non-null sequences, S_i and S_j, if the first N elements of CS_j have the same value, then the following inequality holds for p = 2,3,...N:

$$D_{tw\text{-}lb2}(S_i,CS_j[p:-]) \le D_{tw\text{-}lb}(S_i,CS_j[p:-]) \le D_{tw}(S_i,S_j[p:-])$$

Proof. The proof is given in [13]. ■

6.3. Search algorithm: SimSearch-SST_c

The algorithm SimSearch-ST_c needs to be modified to reflect the fact that there are some suffixes not stored in the index. If SimSearch-ST_c is applied to a SST_C without modification, the subsequences contained in the non-stored suffixes may not be included in the answer set even if similar to a target query sequence. Therefore, the steps of finding and processing the subsequences contained in the non-stored suffixes need to be added to the SimSearch-ST_c.

The proposed algorithm SimSearch-SST_c includes the filtering step and the post-processing step. During the filtering step, $D_{tw\text{-}lb}()$ is used to calculate distances

between Q and the subsequences contained in the stored suffixes, and $D_{tw\text{-}lb2}()$ is used to compute distances between Q and the subsequences contained in the non-stored suffixes. During the post-processing, $D_{tw}()$ is applied to the subsequences included in the candidate answer set. A detail description of the SimSearch-SST_C algorithm is in [13].

6.4. Algorithm analysis

The complexity of SimSearch-SST_C is represented as $O(\frac{(1-r)M\bar{L}^2|Q|}{R_d R_p} + rM\bar{L} + n\bar{L}|Q|)$ where n is the number of subsequences requiring the post-processing, and r is the compaction ratio of a SST_C. Thus, $(1-r)M\bar{L}$ is the number of the stored suffixes, and $rM\bar{L}$ is the number of the non-stored suffixes. Compared with SimSearch-ST_C, SimSearch-SST_C reduces the query processing time by decreasing the number of cumulative distance tables generated during the tree traversal, at the cost of larger n.

7. Experimental results

To study the performance of our similarity search algorithms, we conducted several experiments on stock data sequences extracted from S&P 500 (http://biz.swcp.com/stocks/) and on the artificial data sequences. The stock data were based on their daily closing prices. A total of 545 stock sequences was used with an average length of 232. The expression for generating the artificial sequences was defined as $S_i[p] = S_i[p-1] + Z_p$ where Z_p (p=1,2,…) are independent, identically distributed random variables. Twenty percent of the query sequences were extracted from the stocks whose average prices were below $30, 50% from the stocks whose average prices were between $30 and $60, and 30% from the remaining stocks. The query sequences for the artificial sequences were obtained in a similar manner. The average length of the query sequences was set to 20. All experiments except for the scalability test in Section 7.3 were performed on both the stock and the artificial sequences.

7.1. Index size and query processing time with increasing number of categories

Table 1 shows the sizes of the proposed indices built from the stock sequences, where EL is the equal-length categorization and ME is the maximum-entropy categorization. While the size of ST is not affected by the number of categories, ST_C and SST_C become larger as the number of categories increases. ST_C and SST_C are smaller than ST due to the increased number of common subsequences, and SST_C is smaller than ST_C due to the decreased number of suffixes stored in the index.

Table 1. Index sizes with selected number of categories

# categories	Index Size (Kbytes)				
	ST	ST_C		SST_C	
		EL	ME	EL	ME
10	158,512	5,360	10,534	262	914
20		7,982	15,879	850	2,355
40		12,362	26,069	2,685	7,108
80		18,817	41,288	3,985	18,317
120		26,888	51,942	7,657	28,842
160		32,860	59,927	11,620	37,922
200		37,837	66,357	15,416	45,449
250		43,413	72,937	20,326	53,535
300		48,087	78,297	24,905	60,345

Table 2 shows the average query processing times of the three proposed similarity search algorithms with the average distance-tolerance of 30. On the whole, as the number of categories increases, the executions of SimSearch-ST_C and SimSearch-SST_C become faster. However, their executions slow down when the number of categories exceeds a certain threshold. This threshold value may be used as the optimal number of categories. For example, 200 is the optimal number of categories for SimSearch-SST_C with EL and 120 is for SimSearch-ST_C with ME. Using similar-sized indices, SimSearch-SST_C is faster than SimSearch-ST_C, and SimSearch-SST_C based on ME yields better performance than SimSearch-SST_C based on EL. We obtained similar conclusions from experiments on the artificial sequences.

Table 2. Average query processing times with selected number of categories

# categories	Average Query Processing Time (sec)				
	Sim Search-ST	SimSearch-ST_C		SimSearch-SST_C	
		EL	ME	EL	ME
10	55.30	241.94	84.09	215.73	75.53
20		122.63	35.57	122.75	30.90
40		54.89	25.88	49.61	20.65
80		30.57	21.05	25.90	18.40
120		26.03	20.93	21.30	20.80
160		23.08	21.60	19.13	23.49
200		21.42	22.41	18.63	26.53
250		21.19	23.67	19.08	30.49
300		20.65	25.04	19.55	34.15

7.2. Comparison with sequential scanning

Based on the results from Section 7.1, we chose ME-based SST$_C$ as our index structure and compared its similarity search algorithm with sequential scanning. Table 3 shows their average query processing times with increasing distance-threshold (ε) from 5 to 50. About 50 answers were returned when ε = 5 and about 350,000 answers were reported when ε = 50. Here, SeqScan is sequential scanning and SimSearch-SST$_C$ (k) represents the proposed algorithm with k categories. From Table 1, we know that SST$_C$ with 10, 20, and 80 categories require about 50%, 100%, and 1,000% spaces of database size (1,896 Kbytes), respectively. Our proposed technique is up to 4.2 times faster with 10 categories, 11.1 times faster with 20 categories, and 34.7 times faster with 80 categories than the sequential scanning.

Table 3. Comparison of sequential scanning and our algorithm with selected distance-threshold

distance-threshold(ε)	Query Processing Time (sec)			
	Seq Scan	Sim Search-SST(10)	Sim Search-SST(20)	Sim Search-SST(80)
5	206.04	48.96	18.61	5.94
10	210.48	54.63	21.24	9.01
20	217.14	71.31	27.18	14.15
30	217.45	75.62	30.98	18.49
40	218.13	79.85	34.89	22.71
50	218.96	81.94	38.29	27.08

7.3. Scalability study

To study the scalability of our approach, we compared the execution times of ME-based SimSearch-SST$_C$ with that of sequential scanning by increasing the average length and the number of the artificial sequences. First, we increased the average length of the sequences from 200 to 1,000 while keeping the number of the sequences equal to 200. And we changed the number of sequences from 1000 to 10,000 while maintaining the average length of sequences equal to 200. For both experiments, the numbers of categories were chosen to make the index size smaller than the database size. As shown in Figure 4 and Figure 5, the performance improvements of SimSearch-SST$_C$ is maintained for both long sequences and large number of sequences. Note that the query processing times for both sequential scanning and SimSearch-SST$_C$ increase quadratically with respect to the average sequence length and linearly with respect to the number of sequences.

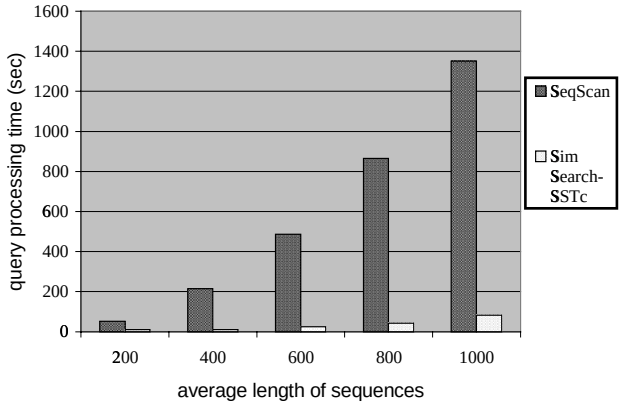


Figure 4. Comparison of sequential scanning and our algorithm with selected length of sequences

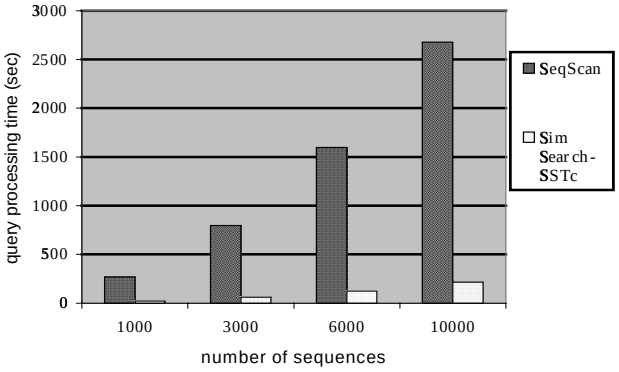


Figure 5. Comparison of sequential scanning and our algorithm with selected number of sequences

8. Conclusion

In this paper we have proposed an indexing method based on a disk-based suffix tree, for fast retrieval of similar subsequences without false dismissals. Because the sampling rates and the lengths of sequences may be different, the proposed method uses a time warping distance as a similarity measure that allows stretching or compressing of sequences along the time axis. Experiments on the stock and the artificial sequences have shown that our proposed technique can be a few orders of magnitude faster than sequential scanning. The contributions of our work are: 1) extending the exact matching algorithm of a suffix tree to similarity searches with a time warping similarity measure, 2) applying the ideas of categorization and sparse suffix tree to make an index structure more compact, and 3) introducing two lower-bound time warping distance functions, $D_{tw\text{-}lb}()$ and $D_{tw\text{-}lb2}()$, to filter out dissimilar subsequences without false dismissals.

The index space can be reduced further if we know the minimum and maximum lengths of the queries. Using a warping window constraint [3], we can calculate the minimum and maximum lengths of the answers. The suffixes whose lengths are shorter than the minimum answer length need not be inserted into the index. For the suffixes whose lengths are longer than the maximum, only the prefixes whose lengths are equal to the maximum length need to be stored in the index.

The subsequences found by similarity searches can be used for predictions, hypothesis testing, clustering and rule discovery. For example, in the medical domain, retrieved subsequences can be used for predicting the disease evolution patterns of a patient; in the web environment, they can be used to discover user web-site visiting patterns.

Our approach can be extended to the sequences of multivariate numeric values. Multivariate values are converted into multi-dimensional cells using multi-dimensional categorization methods such as multiple-attribute type abstraction hierarchy (MTAH) [6]. Then, the same index construction and query processing techniques are applied to the set of converted sequences. We are currently working in this direction for retrieving similar medical image subsequences [7,8].

9. References

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

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

[3] D. J. Berndt and J. Clifford, "Finding Patterns in Time Series: A Dynamic Programming Approach", *Advances in knowledge discovery and data mining, AAAI/MIT Press*, 1996.

[4] P. Bieganski, J. Riedl and J. V. Carlis, "Generalized Suffix Trees for Biological Sequence Data: Applications and Implementation", *Proc. Hawaii International Conference on System Sciences*, 1994.

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

[6] W. W. Chu and K. Chiang, "Abstraction of High Level Concepts from Numerical Values in Databases", *Proc. AAAI Workshop on Knowledge Discovery in Databases*, Seattle, WA, USA, 1994.

[7] W. W. Chu, A. F. Cárdenas, and R. K. Taira. "KMeD: a Knowledge-based Multimedia Medical Distributed Database System", *Information Systems 20(2)*, Premagon-Press/Elsevier Science, 1995.

[8] W. W. Chu, C. Hsu, A. F. Cárdenas, and R. K. Taira, "Knowledge-based Image Retrieval with Spatial and Temporal Constructs", *IEEE TKDE 10(6)*, 1998.

[9] C. Faloutsos and K. Lin, "Fastmap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets", *Proc. ACM SIGMOD*, San Jose, CA, USA, 1995.

[10] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos, "Fast Subsequence Matching in Time-Series Databases", *Proc. ACM SIGMOD*, 1994.

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

[12] J. Kärkkäinen and E. Ukkonen, "Sparse Suffix Trees", *Proc. COCOON*, HongKong, 1996.

[13] S. Park, W. W. Chu, J. Yoon, C. Hsu, "A Suffix Tree for Fast Similarity Searches of Time-warped Subsequences in Sequence Databases", *UCLA-CS-TR-990005*, 1999.

[14] S. Park, D. Lee, W. W. Chu, "Fast Retrieval of Similar Subsequences in Long Sequence Databases", *Proc. IEEE KDEX workshop*, Evanston, IL, 1999

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

[16] D. Rafiei and A. Mendelzon, "Similarity-based Queries for Time Series Data", *Proc. ACM SIGMOD*, Tucson, AZ, 1997.

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

[18] C. E. Shannon and W. Weaver, *The Mathematical Theory of Communication*. University of Illinois Press, 1964.

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

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

[21] N. Yzsdani, M. Özsoyoğlu, "Sequence Matching of Images", *Proc. SSDBM*, Los Alamitos, CA, 1996.

[22] B.-K. Yi, H. V. Jagadish, and C. Faloutsos, "Efficient Retrieval of Similar Time Sequences under Time Warping", *Proc. IEEE ICDE*, 1998.