PERGAMON

Available at www.ComputerScienceWeb.com
POWERED BY SCIENCE @ DIRECT®

Information Systems 28 (2003) 867–883

www.elsevier.com/locate/infosys

Similarity search of time-warped subsequences via a suffix tree☆

Sanghyun Park[a,]*, Wesley W. Chu[b], Jeehee Yoon[c], Jungim Won[c]

[a] *Department of Computer Science and Engineering, Pohang University of Science and Technology (POSTECH), Pohang, South Korea*
[b] *Department of Computer Science, University of California at Los Angeles (UCLA), USA*
[c] *Division of Information and Communication Engineering, Hallym University, South Korea*

Received 21 November 2001; received in revised form 15 October 2002; accepted 15 November 2002

Abstract

This paper proposes an indexing technique for fast retrieval of similar subsequences using the time-warping distance. The time-warping distance is a more suitable similarity measure than the Euclidean distance in many applications where sequences may be of different lengths and/or different sampling rates. The proposed indexing technique employs a disk-based suffix tree as an index structure and uses lower-bound distance functions to filter out dissimilar subsequences without false dismissals. To make the index structure compact and hence accelerate the query processing, it converts sequences in the continuous domain into sequences in the discrete domain and stores only a subset of the suffixes whose first values are different from those of the immediately preceding suffixes. Extensive experiments with real and synthetic data sequences revealed that the proposed approach significantly outperforms the sequential scan and LB scan approaches and scales well in a large volume of sequence databases.
© 2002 Elsevier Science Ltd. All rights reserved.

Keywords: Similarity search; Sequence database; Categorization; Indexing; Suffix tree; Time-warping distance

1. Introduction

A similarity search in sequence databases is an operation that finds sequences or subsequences whose changing patterns are similar to that of a given query sequence [1–3]. Similarity search is of growing importance in many new application domains such as information retrieval, data mining and clustering. Especially in the medical domain, a search for patients with similar disease evolution patterns can augment the process of patient care by providing physicians with insight into the treatment of previous patients with similar medical conditions.

The sequential scan method for similarity search reads each sequence or subsequence sequentially from the database and computes its distance to a query sequence. This method is simple but suffers from severe performance degradation when the database is large. Therefore, an effective indexing scheme is essential as a scalable solution for similarity search.

Most of the previous indexing techniques [1,3,4] for similarity search use the Euclidean distance

☆Recommended by Dr. Nick Koudas.
*Corresponding author.
E-mail address: sanghyun@postech.ac.kr (S. Park).

metric. However, in many applications, the sampling rates and/or 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 [5], this problem has been approached using a similarity measure, called the time-warping distance [5,6], 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 [7], let us consider two sequences, $X = \langle 20, 20, 21, 21, 20, 20, 23, 23 \rangle$ and $Y = \langle 20, 21, 20, 23 \rangle$ where the sequence X is the closing price of a stock taken every day and Y is the closing price of another stock taken every other day. X and Y cannot be compared directly because the sequence X is longer than Y. The Euclidean distance between Y and any subsequence of length four of X is greater than 1.41. However, if we replicate every element of Y using time warping, we find that the two sequences are identical.

It is important to prevent the occurrence of *false dismissals* [1] in similarity search. A false dismissal is defined as missing a part of the final query result. Indexing techniques that assume the *triangular inequality* directly or indirectly may produce false dismissals when the distance function not satisfying the triangular inequality is used as a similarity measure [4]. Unfortunately the time-warping distance does not satisfy the triangular inequality, which can be simply proved by a counter example [4]. This property makes spatial access methods based on the triangular inequality unsuitable for similarity search with the time-warping distance.

In the area of string matching, a suffix tree [8] has been extensively used as an index structure to find the substrings that are exactly matched to a given query string. A suffix tree may be a good candidate for an index structure with the time-warping distance because it does not assume any geometry or any underlying distance functions. However, the following problems have to be addressed so that a suffix tree can be used in similarity search: (1) A suffix tree is designed for exact matching of substrings. Its search algorithm needs to be extended for similarity-based matching of subsequences. (2) A suffix tree is usually built from sequences in the discrete domain; however, sequences we consider in this paper are from the continuous domain. A systematic method to convert continuous values into discrete values is required.

This paper proposes a new indexing technique for the fast retrieval of similar subsequences of different lengths and/or different sampling rates. The proposed technique employs the time-warping distance as a similarity metric and a disk-based suffix tree as an index structure. To reduce the index size, it converts sequences of continuous values into sequences of discrete values and stores only a subset of suffixes whose first values are different from those of the immediately preceding suffixes. When a query sequence Q is submitted, a suffix tree is traversed from the root and the time-warping distances between Q and the subsequences contained in a suffix tree are computed. Because the subsequences contained in a suffix tree are of discrete values, their exact distances to Q cannot be obtained. Therefore, the proposed approach employs lower-bound distance functions to estimate the exact distance without false dismissals.

This paper is organized as follows. Section 2 provides a brief overview of the related work on sequence matching problems and Section 3 gives the definition and property of the time-warping distance. Section 4 introduces the index construction and query processing algorithms for a disk-based suffix tree. The ideas of a categorization and a sparse suffix tree are applied to the similarity search algorithms in Sections 5 and 6, respectively. And, Section 7 compares the proposed algorithm with the sequential scan and LB scan algorithms.

2. Related work

There has been much research on similarity search in sequence databases. Agrawal et al. [1] proposed the *F-Index*, a similarity searching technique for whole sequence matching. Sequences are converted into the frequency domain by the Discrete Fourier Transform (DFT) and are subsequently mapped into multi-dimensional points that are managed by an R*-tree [9]; this technique was extended to locate similar subsequences [3]. Since both approaches use the Euclidean distance, sequences of different sampling rates cannot be matched.

Several sequence matching techniques that allow transformations or noises were proposed. Goldin et al. [10] grouped sequences into equivalent classes using the normal form. Although the normal form is invariant to shape-based transformations such as scaling and shifting, it does not handle the compressions or the stretches of element values along the time axis. Rafiei et al. [7] proposed a class of sequence transformations that can be used in a query language to express similarity with an R-tree [11] index. The proposed transformations handle moving average and global time scaling, but not time-warping. Agrawal et al. [2] proposed a new model of similarity that captures the intuitive notion that two sequences should be considered similar if they have enough non-overlapping time ordered pairs of similar subsequences.

More recent approaches permit the matching of sequences of different lengths. Bozcaya et al. [12] presented a modified version of an edit distance, judging that two sequences are similar if a majority of their elements match. Yi et al. [4] supported the time-warping distance using a two-step filtering process: a FastMap [13] index filter followed by a lower-bound distance filter. The underlying index structures of both approaches [12,4] are based on the triangular inequality. The approach suggested by Keogh et al. [14,15] read a data sequence sequentially from the database, converted it into an ordered list of piece-wise linear segments using the best fitting line, and applied the modified time-warping distance measure. Park et al. [16] proposed a segment-based subsequence searching scheme for the database of long sequences. This scheme changed the similarity measure from the time-warping distance to the piece-wise time-warping distance and limited the number of data subsequences to be compared with a query sequence. Although the above approaches [14–16] based on piece-wise linear segments significantly reduce the search time, they may miss the similar subsequences starting or ending at the middle of segments.

Several shape-based similarity matching schemes were proposed. Agrawal et al. [17] demonstrated a shape definition language (SDL) and provided an index structure for speed up the execution of SDL queries. Shatkay et al. [18] introduced 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 are not suitable for applications that care about specific element values.

There are also a few approaches for the matching of biological sequences. Bieganski et al. [19] proposed to use a disk-based suffix tree for solving the sequence alignment problem, and Wang et al. [20] addressed the problem of discovering patterns in protein databases with a string edit distance. Whereas we focus on the sequences of continuous numeric values, both approaches center on the sequences of discrete symbols. Furthermore, the approach of Wang et al. [20] uses a memory resident suffix tree, making it infeasible for a large sequence database.

3. Time-warping distance

Finding a similarity measure for sequences is not easy because sequences that are qualitatively the same may be different quantitatively. 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.

This paper uses a time-warping similarity measure [5,6] that allows sequences to be stretched or compressed along the time axis. Time warping is a generalization of classical algorithms for comparing discrete sequences to sequences of continuous values. Time warping is extensively used in matching of voice, audio and medical signals (electrocardiograms). To find the minimum difference between two sequences, time warping maps each element of a sequence to one or more neighboring elements of another sequence.

Let us introduce a few notations before presenting a formal definition of the time-warping distance. We use the notation $X = \langle X[1], \ldots, X[n] \rangle$ for a sequence with n elements. $X[i]$ denotes the ith element of X and $|X|$ denotes the number of elements in X. $X[i:j]$ is a subsequence of X containing elements in positions i through j. $X[i:-]$ is a subsequence of X starting at the ith element position and ending at the last element position. That is, $X[i:-] = X[i:|X|]$. $X[i:-]$ also represents the suffix of X starting at the ith position. Let us now give the formal definition of the time-warping distance.

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

$$D_{tw}(X, Y) = \sqrt[p]{D_{base}(X[1], Y[1])^p + MinStutterDist(X, Y)^p},$$

$$D_{base}(X[1], Y[1]) = |X[1] - Y[1]|,$$

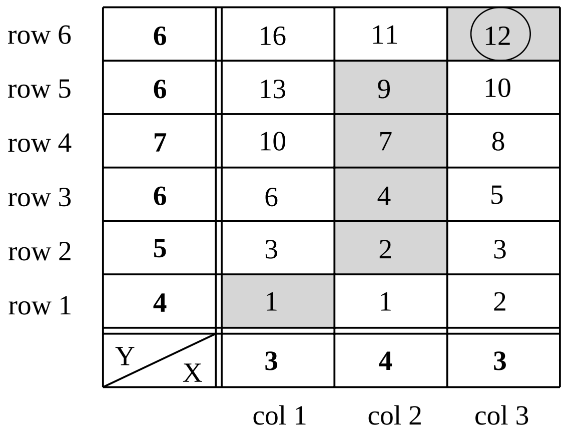
$$MinStutterDist(X, Y) = min \begin{cases} D_{tw}(X, Y[2:-]), \\ D_{tw}(X[2:-], Y), \\ D_{tw}(X[2:-], Y[2:-]). \end{cases}$$

Definition 1 generalizes the original time-warping distance [4–6] to handle various values of p. It is called the L_1-based time-warping distance when $p = 1$, the L_2-based time-warping distance when $p = 2$, and the L_∞-based time-warping distance when $p = \infty$. Berndt et al. [6] and Yi et al. [4] focused on the L_1-based time-warping distance, and Park et al. [21] mainly handled the L_∞-based time-warping distance. However, this paper considers all possible values of p.

$D_{tw}(X, Y)$ can be efficiently calculated using a dynamic programming technique [6] based on the recurrence relation. The dynamic programming algorithm [6] fills in table T, which contains cumulative distances, as the computation proceeds. The final cumulative distance, $T[|Y|][|X|]$, is the minimum distance between X and Y, and the matching of elements can be traced backward in the table by choosing the previous cells with the lowest cumulative distance. This distance computation has the complexity $O(|X||Y|)$. Fig. 1 shows the cumulative distance table for computing the L_1-based time-warping distance between the two sequences $X = \langle 3, 4, 3 \rangle$ and $Y = \langle 4, 5, 6, 7, 6, 6 \rangle$. Here $D_{tw}(X, Y) = 12$ because $T[|Y|][|X|] = T[6][3] = 12$.

The distance between X and any prefix of Y can be computed simply by reading the rightmost column of each row of the cumulative distance table. More specifically, the distance between X and $Y[1:j]$ $(j = 1, 2, \ldots, |Y|)$ is stored in the rightmost column of the jth row. In the above example, $D_{tw}(X, Y[1:4])$ is 8, as seen in the rightmost column of the row 4. Thus the determination as to whether the time-warping distance of two sequences is greater than a distance threshold ε does not require us to build the entire cumulative distance table, as proven by the following theorem.

Theorem 1. *If all the columns of the top row of the cumulative distance table have values greater than a distance tolerance* ε*, adding more rows on this table does not yield a new value less than or equal to* ε.



row 6	**6**	16	11	12
row 5	**6**	13	9	10
row 4	**7**	10	7	8
row 3	**6**	6	4	5
row 2	**5**	3	2	3
row 1	**4**	1	1	2
	Y \ X	**3**	**4**	**3**
		col 1	col 2	col 3

Fig. 1. Cumulative distance table for computing the L_1-based time-warping distance between $X = \langle 3, 4, 3 \rangle$ and $Y = \langle 4, 5, 6, 7, 6, 6 \rangle$. $D_{tw}(X, Y) = 12$ because $T[|Y|][|X|] = T[6][3] = 12$.

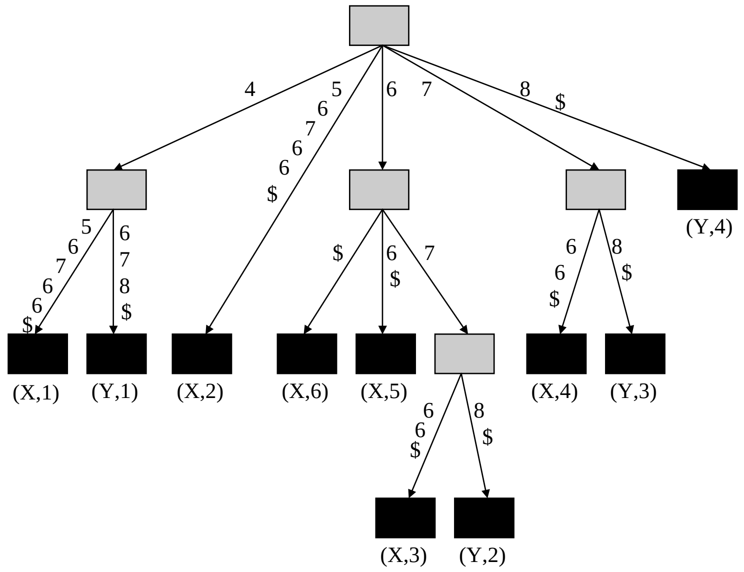


Fig. 2. The suffix tree constructed from $X = \langle 4,5,6,7,6,6 \rangle$ and $Y = \langle 4,6,7,8 \rangle$. The six suffixes ($\langle 4,5,6,7,6,6 \rangle$, $\langle 5,6,7,6,6 \rangle$, $\langle 6,7,6,6 \rangle$, $\langle 7,6,6 \rangle$, $\langle 6,6 \rangle$, $\langle 6 \rangle$) from X and the four suffixes ($\langle 4,6,7,8 \rangle$, $\langle 6,7,8 \rangle$, $\langle 7,8 \rangle$, $\langle 8 \rangle$) from Y are extracted and then inserted into the suffix tree. \$ denotes an end marker of a suffix.

Proof. The proof is given in [22]. □

Let us consider Fig. 1 again. If ε is 3, after inspecting row 3, we can determine that the distance between X and Y is greater than ε because all the columns of row 3 have values greater than 3. Therefore, we do not have to fill the remaining three rows. In the remaining sections, we use Theorem 1 to reduce the search space of an index structure.

4. Similarity search using a suffix tree

This section proposes to use a suffix tree as an index structure for similarity search with the time-warping distance. Before describing the methods for constructing and traversing a suffix tree, we present the definition and internal structure of a suffix tree.

A *trie* is a data structure used for indexing a set of keywords. A *suffix trie* [8] is a trie whose set of keywords comprises the suffixes of sequences. Nodes with a single outgoing edge can be collapsed, yielding the structure known as a *suffix tree* [8]. A suffix tree is generalized [19,20] to allow multiple sequences to be stored in the same tree. Each suffix of a sequence is represented by a leaf node. More specifically, $X[i:-]$ is expressed by a leaf node labeled with $(ID(X), i)$, where $ID(X)$ is the identifier of X and i is the offset from which the suffix starts. We use the notation $label(N_1, N_2)$ for the concatenated labels on the path from a node N_1 to another node N_2. Fig. 2 shows the suffix tree constructed from the two sequences $X = \langle 4,5,6,7,6,6 \rangle$ and $Y = \langle 4,6,7,8 \rangle$. The six suffixes ($\langle 4,5,6,7,6,6 \rangle$, $\langle 5,6,7,6,6 \rangle$, $\langle 6,7,6,6 \rangle$, $\langle 7,6,6 \rangle$, $\langle 6,6 \rangle$, $\langle 6 \rangle$) from X and the four suffixes ($\langle 4,6,7,8 \rangle$, $\langle 6,7,8 \rangle$, $\langle 7,8 \rangle$, $\langle 8 \rangle$) from Y are extracted and then inserted into the suffix tree. \$ denotes an end marker of a suffix.

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 construction algorithm is applied to the concatenated sequence. A suffix tree being created by this process has to reside in main memory during construction. Therefore, this approach is not feasible in a large sequence set.

To remedy this problem, we use an incremental disk-based suffix tree construction method proposed in [19]. Two suffix trees, representing two disjoint sets of sequences, are merged to produce a single suffix tree by performing the pre-order traversal on both 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.

Two suffix trees, one for X and another for Y, are merged with the time complexity $O(|X| + |Y|)$. Therefore, the construction of a suffix tree from m data sequences, whose average length is $\bar{L}$, requires the complexity $O(m\bar{L})$. 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 nodes is at least 2. As a result, the maximum number of nodes and overall space requirement of a suffix tree are linear to $m\bar{L}$ [8].

4.2. Search algorithm: SimSearch-ST

A suffix tree is a useful index structure for exact matching of subsequences. To find the subsequences exactly matched to a query sequence Q, the suffix tree is traversed from the root and the traversal is terminated when the end of Q is reached or a node is reached beyond which further traversal is not possible. This traversal algorithm is performed in $O(|Q|)$. Although this algorithm is simple and fast, it cannot be directly applied to the similarity search problem.

Problem Definition. Given a set of data sequences of arbitrary lengths, a query sequence Q, and a distance tolerance ε, find those subsequences whose time-warping distances to Q are less than or equal to ε.

Additional types of queries include the nearest neighbor queries (e.g., “find the five subsequences most similar to a given query sequence”) and the “all pairs” queries (e.g., “report all pairs of subsequences that are within distance ε from each other”). Both types of queries can be handled by our approach using a branch-and-bound algorithm [23] together with a spatial join algorithm [24].

The proposed similarity search algorithm SimSearch-ST is given in Algorithm 1. The search starts from the root and continues the depth-first traversal until all the subsequences whose time-warping distances to Q are within ε are found.

Algorithm 1: Similarity search algorithm SimSearch-ST

```
Input: root node $R$, query sequence $Q$, distance
       tolerance $\varepsilon$
Output: answerSet
cumDistTable $\leftarrow$ NULL;
answerSet $\leftarrow$ Filter-ST($R, Q, \varepsilon$, cumDistTable);
return answerSet;
```

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. Simply we assume that every edge connecting a node N and its child node CN_i is labeled with a single symbol.

To find a new answer, Filter-ST builds a cumulative distance table for Q and $label(N, CN_i)$. If N is the root (i.e., CN_i is the direct child of the root), then the distance table is built from the bottom. Otherwise, the distance table is constructed by appending a new row on the existing table T which has been accumulated from the root to N. The algorithm calls the function AddRow($T, Q, label(N, CN_i), D_{tw}$) to build a new cumulative distance table, using the distance function D_{tw}, by appending a new row for $label(N, CN_i)$ on T. If the rightmost column of a newly added row has a value less than or equal to a distance tolerance ε, then $label(root, CN_i)$ is added into the answer set.

To determine if visiting the subtree of CN_i is needed, the algorithm reads each column of a newly added row. If at least one column has a value less than or equal to ε, then the algorithm continues down the tree to find more answers. Otherwise, the algorithm moves to the next child of N. This branch-pruning method is based on the Theorem 1 shown in Section 3.

Algorithm 2: Filtering algorithm Filter-ST

```
Input  : node $N$, query sequence $Q$, distance tolerance $\varepsilon$, 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 rightmost column value of a newly added row;
    Let $minDist$ be the minimum column value of a newly added row;
    if $dist \le \varepsilon$ then
        answerSet ← answerSet ∪ {label(GetRoot($CN_i$), $CN_i$)};
    if $minDist \le \varepsilon$ then
        answerSet ← answerSet ∪ Filter-ST($CN_i$, $Q$, $\varepsilon$, $CT_i$);
return answerSet;
```

4.3. *Algorithm complexity*

Before analyzing the complexity of SimSearch-ST, let us examine the complexity of the sequential scan method. The sequential scan method 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 data 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 the sequential scan is expressed as $O(m\bar{L}^2|Q|)$.

SimSearch-ST is computationally less expensive than the sequential scan because (1) the branch-pruning method reduces the search space and (2) the suffixes with common prefixes share the cumulative distance tables during index traversal. Thus, SimSearch-ST has the complexity $O(m\bar{L}^2|Q|/R_dR_p)$, where $R_d(\geqslant 1)$ is the reduction factor saved by sharing the cumulative distance tables, and $R_p(\geqslant 1)$ is the reduction factor gained from the branch pruning.

R_d grows as the length and the number of common edges of a suffix tree increase. Given k suffixes, $s_1, \ldots, s_k$, whose first t elements are the same, the construction of k cumulative distance tables requires the computation of $|s_1||Q| + \cdots + |s_k||Q|$ cells. However, it is reduced to $t|Q| + (|s_1| - t)|Q| + \cdots + (|s_k| - t)|Q|$ 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 the following:

$$R_d = \frac{|s_1| + \cdots + |s_k|}{(|s_1| + \cdots + |s_k|) - (k-1)t}.$$

While R_d relies upon the distribution of element values, R_p is dependent on a distance tolerance ε specified by the user. That is, R_p increases as ε decreases. If ε is so small that just one or two subsequences can be answers, only the topmost part of the index may be visited. In another extreme case where ε is large enough for all subsequences to be answers, every node of the index needs 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 the sequential scan.

5. Similarity search using categorization

This section introduces the concept of categorization to decrease the number of possible values that elements can take, hence increasing the length and the number of common subsequences. As explained in the previous section, the index size and the query processing time reduce as the length and the number of common subsequences increase.

To get the categorized representation of element values, we first generate the set of categories and determine their ranges. Then we convert every element value into the symbol of the corresponding category. For example, given two categories $C_1 = [0.1, 3.9]$ and $C_2 = [4.0, 10.0]$, the sequence $Y = \langle 5.27, 2.56, 3.85 \rangle$ is transformed to $Y^C = \langle C_2, C_1, C_1 \rangle$ where Y^C denotes the sequence obtained from Y by categorization. After converting elements into discrete symbols, we build a suffix tree from a set of symbol sequences. The resultant tree is called a Suffix Tree with Categorization (STC). It is also constructed by performing a series of binary merges.

5.1. Categorization method

We take the maximum-entropy categorization method for its simple implementation, albeit other categorization approaches like the type abstraction hierarchy (TAH) [25] and the k-means algorithm may also be used. The entropy [26] of a categorization is defined as: $H = -\sum_{i=1}^{k} P(C_i)\log(P(C_i))$ where k is the number of categories given as an input parameter and $P(C_i)$ is the probability that an element is included in the ith category. To minimize the loss of the information about 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 $(P(C_1) = P(C_2) = \cdots = P(C_k))$.

5.2. Modified distance function: D_{tw-lb}

Whereas the edges of a suffix tree are labeled with numeric values, the edges of an STC are labeled with symbols. As a result, the exact time-warping distance between a query sequence and any subsequence contained in an STC cannot be computed. Therefore, we introduce a new distance function D_{tw-lb} that returns a lower-bound distance of D_{tw}.

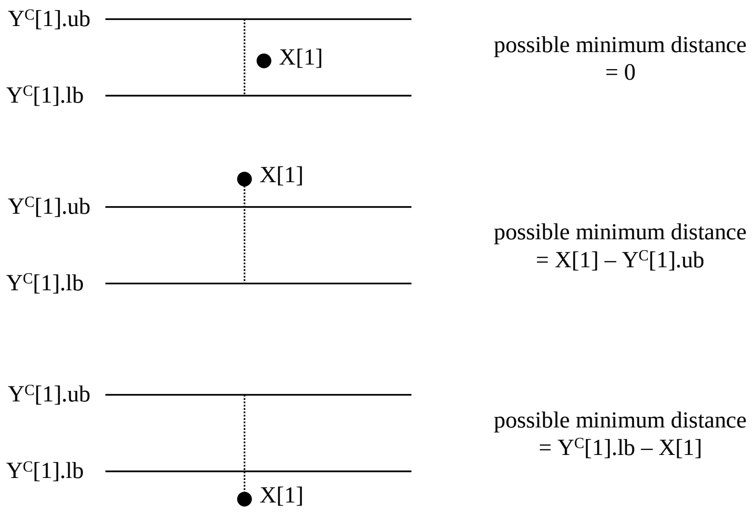
Definition 2. Given any two non-null sequences X and Y, the distance function $D_{tw-lb}(X, Y^C)$ that returns a lower-bound distance of $D_{tw}(X, Y)$ is defined as follows:

$$D_{tw-lb}(X, Y^C) = \sqrt[p]{D_{base-lb}(X[1], Y^C[1])^p + MinStutterDist(X, Y^C)^p},$$

$$D_{base-lb}(X[1], Y^C[1]) = \begin{cases} 0 & \text{if } Y^C[1].lb \leqslant X[1] \leqslant Y^C[1].ub \\ X[1] - Y^C[1].ub & \text{if } X[1\, Y^C[1].ub, \\ Y^C[1].lb - X[1] & \text{if } X[1] < Y^C[1].lb, \end{cases}$$

$$MinStutterDist(X, Y^C) = min \begin{cases} D_{tw-lb}(X, Y^C[2:-]), \\ D_{tw-lb}(X[2:-], Y^C), \\ D_{tw-lb}(X[2:-], Y^C[2:-]). \end{cases}$$

In Definition 2, $Y^C[1].lb$ and $Y^C[1].ub$ express the range of the category denoted by the symbol $Y^C[1]$. As shown in Fig. 3, $D_{tw-lb}(X[1], Y^C[1])$ returns the possible minimum distance between $X[1]$ and $Y^C[1]$.



Y^C[1].ub
X[1]
Y^C[1].lb
possible minimum distance = 0
X[1]
Y^C[1].ub
Y^C[1].lb
possible minimum distance = X[1] – Y^C[1].ub
Y^C[1].ub
Y^C[1].lb
X[1]
possible minimum distance = Y^C[1].lb – X[1]

Fig. 3. Minimum distances between $X[1]$ and $Y^C[1]$. $Y^C[1].lb$ and $Y^C[1].ub$ express the range of the category denoted by the symbol $Y^C[1]$.

To guarantee no false dismissal, the distance returned by $D_{tw-lb}(X, Y^C)$ should always be less than or equal to the distance computed by $D_{tw}(X, Y)$, as stated by the following theorem.

Theorem 2. *For any two non-null sequences X and Y, the following inequality holds:*

$D_{tw-lb}(X, Y^C) \leqslant D_{tw}(X, Y)$.

Proof. The proof is shown in [22]. □

5.3. *Search algorithm:* SimSearch-STC

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

Algorithm 3: Similarity search algorithm SimSearch-STC

```
Input: root node $R$, query sequence $Q$, distance tolerance $\varepsilon$
Output: answerSet
cumDistTable ← NULL;
candidateSet ← Filter-STC ($R$, $Q$, $\varepsilon$, cumDistTable);
answerSet ← PostProcess (candidateSet, $Q$, $\varepsilon$);
return answerSet;
```

To find the candidate subsequences whose lower-bound time-warping distances to a query sequence Q are within ε, Filter-STC is called recursively. Filter-STC is the same as Filter-ST except that the former uses D_{tw-lb} to build a cumulative distance table while the latter uses D_{tw}. Since a lower-bound time-warping distance is used for filtering, the dissimilar subsequences whose actual time-warping distances to Q are larger than ε may be included in the candidate answer set. These subsequences are called *false alarms* [1,3]. To detect and discard false alarms, PostProcess retrieves the actual data subsequences corresponding to each candidate answer and computes their time-warping distances using D_{tw}.

The complexity of SimSearch-STC is represented as $O(m\bar{L}^2|Q|/R_dR_p + n\bar{L}|Q|)$ where n is the number of subsequences requiring the post-processing. Hence, the left expression represents the cost for filtering and the right expression for post-processing. Compared to the algorithm SimSearch-ST, SimSearch-STC has a larger value of R_d at the expense of the cost for post-processing.

6. Similarity search with sparse suffix tree

A suffix tree that stores only a subset of suffixes is called a sparse suffix tree [27]. Since the size of a suffix tree is linear in the number of leaf nodes, 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 $(0 \leqslant r < 1)$ that is defined as follows:

$$r = \frac{\text{the number of non-stored suffixes}}{\text{the number of all suffixes}}.$$

The value of r is highly dependent on the number of distinct values that elements can take. This section proposes an indexing technique SimSearch-SSTC that uses a Sparse Suffix Tree With Categorization (SSTC) as an index structure to reduce the index size and accelerate the query processing.

6.1. *Index construction*

Similar to STC, an SSTC is built from sequences of symbols. However, unlike STC, only suffixes whose first values are different from those of the immediately preceding suffixes are stored in an SSTC. That is, $Y^C[i:-]$ is stored in a SSTC only if $Y^C[i] \neq Y^C[i-1]$. For example, given $Y^C = \langle A, A, A, C, B, B \rangle$, only three suffixes, $Y^C[1:-]$, $Y^C[4:-]$, and $Y^C[5:-]$, are stored in an SSTC. Therefore, the compaction ratio $r = \frac{3}{6} = 0.5$.

6.2. *Modified distance function:* D_{tw-lb2}

While we can get the distance between X and any prefix of Y^C by reading the rightmost columns of the cumulative distance table for X and Y^C, there is no direct way to compute the distance between X and any suffix of Y^C except for building a new distance table. However, if the first t elements of Y^C have the same value, we can obtain a lower-bound distance of $D_{tw-lb2}(X, Y^C[i:-])$ $(i = 1, \ldots, t)$ using a new distance function $D_{tw-lb2}(X, Y^C[i:-])$.

Definition 3. For any two non-null sequences X and Y^C, if the first t elements of Y^C have the same value, then the distance function $D_{tw-lb2}(X, Y^C[i:-])$ $(i = 1, \ldots, t)$ that returns a lower-bound distance of $D_{tw-lb}(X, Y^C[i:-])$ is defined as follows:

$$D_{tw-lb2}(X, Y^C[i:-]) = D_{tw-lb}(X, Y^C) - (i-1) * D_{base-lb}(X[1], Y^C[1]).$$

If we know the value of $D_{tw-lb}(X, Y^C)$, then $D_{tw-lb2}(X, Y^C[i:-])$ can be computed with complexity $O(1)$. The distance returned from $D_{tw-lb2}(X, Y^C[i:-])$ is always less than or equal to $D_{tw-lb}(X, Y^C[i:-])$, as stated in the following theorem.

Theorem 3. *For any two non-null sequences X and Y^C, if the first t elements of Y^C have the same value, then the following inequality holds for $i = 1, \ldots, t$:*

$$D_{tw-lb2}(X, Y^C[i:-]) \leqslant D_{tw-lb}(X, Y^C[i:-]).$$

Proof. The proof is shown in [22]. □

6.3. Search algorithm: SimSearch – SSTC

The proposed similarity search algorithm SimSearch-SSTC consists of the filtering stage and the post-processing stage. To filter out dissimilar subsequences, the filtering algorithm Filter-SSTC is called recursively. Filter-SSTC uses D_{tw-lb} for calculating the distances between Q and the subsequences contained in *stored suffixes*, and D_{tw-lb2} for computing the distances between Q and the subsequences contained in *non-stored suffixes*. The procedures for finding candidates and pruning branches are identical to those of Filter-STC. During the post-processing, D_{tw} is applied to the subsequences corresponding to each candidate. A detailed description of the algorithm SimSearch-SSTC is in [22].

The complexity of SimSearch-SSTC is $O((1-r)m\bar{L}^2|Q|/R_dR_p + rm\bar{L} + n\bar{L}|Q|)$ where n is the number of subsequences requiring the post-processing and r is the compaction ratio of the index. Thus $(1-r)m\bar{L}$ is the number of stored suffixes, and $rm\bar{L}$ is the number of non-stored suffixes. Comparing with SimSearch-STC, SimSearch-SSTC tries to improve the performance by reducing the number of cumulative distance tables generated during the tree traversal, at the cost of larger n.

7. Experimental evaluation

To study the performance and scalability of the proposed similarity search algorithms, we conducted extensive experiments with real and synthetic data sets. This section first describes the evaluation environment and then chooses the one from the proposed similarity search algorithms after comparing them in terms of space and time efficiency. We then compare the performance and scalability with the previous approaches.

7.1. Evaluation environment

For performance evaluation, we implemented the proposed algorithms in C and C + + programming languages and then compared them with the previous approaches using stock, electrocardiogram, and random-walk data sets. All the experiments were performed under Solaris 7 operating system.

7.1.1. Approaches to be compared

The sequential scan [6] and the LB-scan [4] were selected as candidate algorithms, because they use the time-warping distance metric and support the subsequence search without false dismissals. The approaches using the piece-wise linear segments [14–16] were excluded due to the possibility of false dismissals.

(1) **The sequential scan method** [6] reads every sequence from the database and applies a dynamic programming technique to build a cumulative distance table. The number of cumulative distance tables to be generated is the same as the number of suffixes contained in a data sequence. This method does not require post-processing.
(2) **The LB-Scan method** [4] reads every sequence from the database and computes a lower-bound distance function D_{lb} to discard non-qualifying sequences without false dismissals. This method reduces the CPU cost from $O(|X||Y|)$ to $O(|X|+|Y|)$ because the computation of $D_{lb}(X, Y)$ is linear to both $|X|$ and $|Y|$. Since a lower-bound distance function is used for filtering, post-processing is required.
(3) **Proposed methods** perform a series of binary merges to build an index. Each node of a suffix tree stores the location of its first right sibling in addition to the locations of its children. As a result, only the nodes on the path from the root to the node being inspected need to reside in main memory during the index traversal. Whereas SimSearch-ST does not require post-processing, SimSearch-STC and SimSearch-SSTC require post-processing.

7.1.2. Data set

We used one synthetic and two real data sets for the experiments. The S&P 500 stock data sequences (http://biz.swcp.com/stocks) were based on their daily closing prices. The number of and the average length of data sequences were 545 and 232, respectively. The size of this data set was 851 kbytes.

The electrocardiogram data sequences were also used in the experiments. An electrocardiogram is a recording of the heart which allows for doctors to evaluate the patient's cardiac condition such as irregular heart beats and rhythm. The number and the average length of electrocardiogram data sequences were 340 and 659, respectively. The size of this data set was 2155 kbytes.

The random-walk data sequences were used for scalability testing. The expression generating these data sequences was defined as $X[i] = X[i-1] + Z_i$ where Z_i $(i = 1, 2, \ldots)$ are independent, identically distributed, random variables. The number of random-walk data sequences and their average length were determined by the purpose of each scalability testing.

One hundred query sequences were generated by the following procedure from each data set: (1) select a random sequence from the database; (2) extract a random subsequence from that data sequence; (3) take a random value from an appropriate range[1] for every element of the subsequence; and (4) add the value to its corresponding element.

7.1.3. System configuration

The hardware platform for the experiments was the Sun UltraSparc-10 workstation equipped with 333 MHz UltraSparc-IIi CPU, 512 MB RAM, and 26 GB SCSI hard disk with 10,000 RPM. The operating system used was Solaris 7.

7.2. Space and time efficiency of the proposed algorithms

Using the S&P stock data sequences, we compared the space and time efficiency of the proposed algorithms. Fig. 4 shows the size of the proposed indices. Whereas the size of a suffix tree (ST) is independent of the number of categories, the size of STC and SSTC becomes larger as the number of categories increases. STC and SSTC are smaller than ST due to the increased number of common subsequences, and SSTC is smaller than STC due to the decreased number of suffixes stored in the index.

Fig. 5 shows the average query processing time of the proposed search algorithms with a selected number of categories. The similarity metrics were the L_1-based, the L_2-based and the L_∞-based time-warping distances. A distance tolerance was adjusted for each query to retrieve $10^{-3}\%$ of the total number of data subsequences. SimSearch-STC and SimSearch-SSTC speed up on the whole as the number of categories increases. However, their executions slow down when the number of

[1] The range is $[-std/10, std/10]$, where std is the standard deviation of the sequence.

#categories	Index Size (Kbytes)		
	ST	STC	SSTC
10	157,230	10,459	853
20		15,732	2,248
40		25,778	6,851
80		40,982	18,038
120		51,511	28,409
160		59,512	37,466
200		65,859	44,988
250		72,484	53,065
300		77,652	59,723

Fig. 4. The size of the proposed indices with selected number of categories. Stock data sequences were used for this experiment.

categories exceeds a certain threshold. This threshold value may be regarded as the optimal number of categories. For example, when the distance function is the L_1-based time-warping distance, 250 is the optimal number of categories for SimSearch-STC and 80 is for SimSearch-SSTC.

Let us first consider the average query processing time with the L_1-based time-warping distance metric. SimSearch-STC and SimSearch-SSTC are slower than SimSearch-ST when the number of categories is small (i.e. $\leqslant 20$). This is because the lower-bound distance functions used in SimSearch-STC and SimSearch-SSTC lose their tightness to the original distance function when the number of categories is too few. However, SimSearch-STC and SimSearch-SSTC become faster than SimSearch-ST when the number of categories exceeds 20. SimSearch-STC performs slightly better than SimSearch-SSTC under the same number of categories. We obtained similar results with the L_2-based and the L_∞-based time-warping distances.

These experiments show that (1) SSTC is the most space-effective approach, and (2) SimSearch-STC is the most time-effective approach unless the number of categories is too few. With all these results, we decided SimSearch-SSTC as the best one because it solves the size problem of a suffix tree while preserving the good performance. The next section compares SimSearch-SSTC with the previous similarity search algorithms using the stock and electrocardiogram data sets.

7.3. Performance comparisons

According to the results of Section 7.2, we chose SimSearch-SSTC and compared it with the LB-Scan and the sequential scan. Figs. 6–11 shows their average query processing time with the increasing number of answers (i.e., increasing values of a distance tolerance). The number of categories used by our approach was 20.

#categories	Average Query Processing Time (sec)								
	SimSearch-ST			SimSearch-STC			SimSearch-SSTC		
	L_1	L_2	L_{inf}	L_1	L_2	L_{inf}	L_1	L_2	L_{inf}
10	10.6	4.7	2.9	24.6	22.5	22.6	38.9	45.3	17.2
20				12.0	9.6	9.1	15.5	17.7	6.0
40				7.3	4.5	3.7	8.1	8.0	2.6
80				4.8	3.4	2.4	6.2	5.2	2.1
120				4.4	2.7	1.6	6.9	5.5	2.1
160				4.3	2.6	1.6	8.6	6.3	2.5
200				4.3	2.6	1.4	10.4	7.4	2.6
250				4.1	2.6	1.6	12.9	9.2	3.1
300				4.6	2.9	1.7	16.0	10.1	3.2

Fig. 5. The average query processing time of the proposed algorithms with selected number of categories. Stock data sequences were used for this experiment. A distance tolerance was adjusted for each query to retrieve $10^{-3}\%$ of the total number of data subsequences.

Figs. 6 and 7 used the L_1-based time-warping distance metric with the stock and electrocardiogram data sets, respectively. It is clear from the graphs that our approach consistently outperforms both approaches. With the stock data set, our approach is faster than the sequential scan by up to 18 times and the LB-scan by up to 3 times. With the electrocardiogram data set, our approach is faster than the sequential scan by up to 17 times and the LB-scan by up to 6 times. Notice that the performance benefit of our approach over the LB scan grows larger on both data sets as the number of answers increases.

Figs. 8 and 9 used the L_2-based time-warping distance metric with the stock and electrocardiogram data sets, respectively. As shown in the figures, our approaches performs faster than both competitors. With the stock data set, our approach is faster than the sequential scan by up to 13 times but the performance benefit over the LB-scan is not conspicuous. With the electrocardiogram data set, our approach is faster than the sequential scan by up to 21 times and the LB scan by up to 4.5 times.

Figs. 10 and 11 used the L_∞-based time-warping distance metric with the stock and electrocardiogram data sets, respectively. We can easily see from the figures that our approach beats the previous two approaches. With the stock data set, our approach is faster than the sequential scan by up to 40 times and the LB scan by up to 3.4 times. With the electrocardiogram data sequences, our

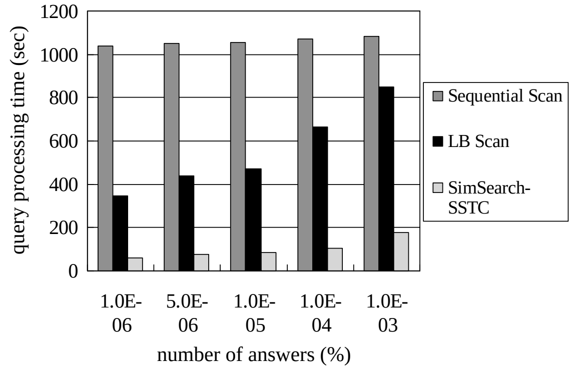


Fig. 7. Average query processing time with the increasing number of answers. The L_1-based time-warping distance was used on the electrocardiogram data sequences.

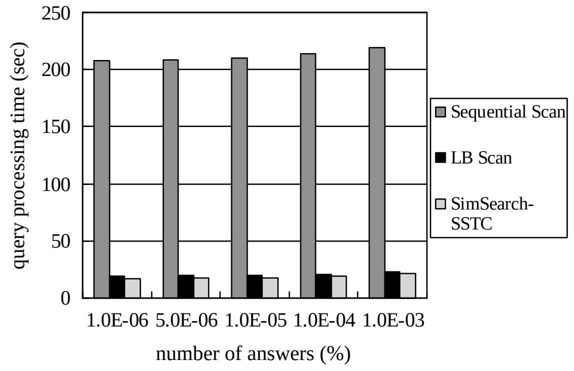


Fig. 8. Average query processing time with the increasing number of answers. The L_2-based time-warping distance was used on the stock data sequences.

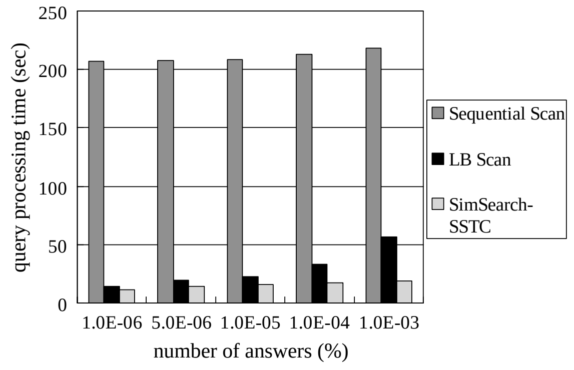


Fig. 6. Average query processing time with the increasing number of answers. The L_1-based time-warping distance was used on the stock data sequences.

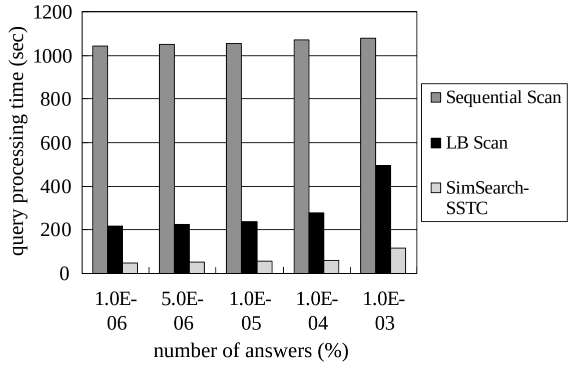


Fig. 9. Average query processing time with the increasing number of answers. The L_2-based time-warping distance was used on the electrocardiogram data sequences.

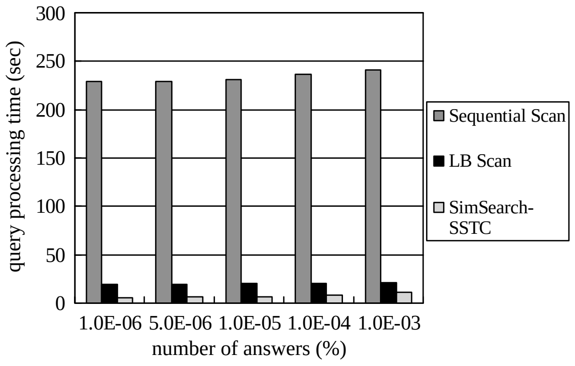


Fig. 10. Average query processing time with the increasing number of answers. The L_∞-based time-warping distance was used on the stock data sequences.

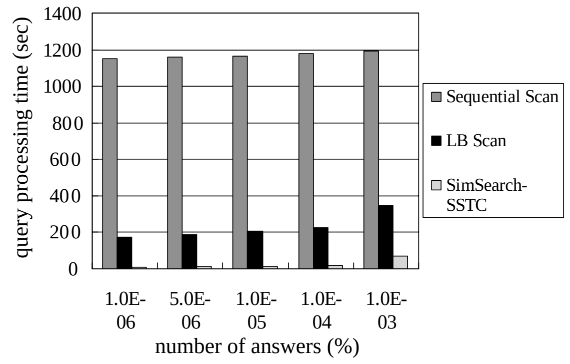


Fig. 11. Average query processing time with the increasing number of answers. The L_∞-based time-warping distance was used on the electrocardiogram data sequences.

approach is faster than the sequential scan by up to 108 times and the LB-scan by up to 16 times.

7.4. Scalability comparisons

To study the scalability of our approaches, we compared the average query processing time of SimSearch-SSTC with that of the sequential scan and the LB scan by increasing the average length and the number of random walk data sequences.

We first increased the average length of data sequences from 200 to 1000 while keeping the number of data sequences equal to 200. Then we changed the number of data sequences from 1000 to 10,000 while maintaining their average length equal to 200. The number of categories for both experiments was 20 and the distance tolerance was adjusted to retrieve 10^{-3}% of the total number of data subsequences.

Figs. 12, 14, and 16 show the average query processing time of the three methods with the increasing average length of data sequences. They used the L_1-based, the L_2-based and the L_∞-based time-warping distances, respectively. As seen from these figures, the performance improvement of SimSearch-SSTC is maintained with any value of p in the L_p-based time-warping distance metric even when sequences are very long.

Figs. 13–17 show the average query processing time of the three methods with the increasing number of data sequences. They used the L_1-based, the L_2-based and the L_∞-based time-

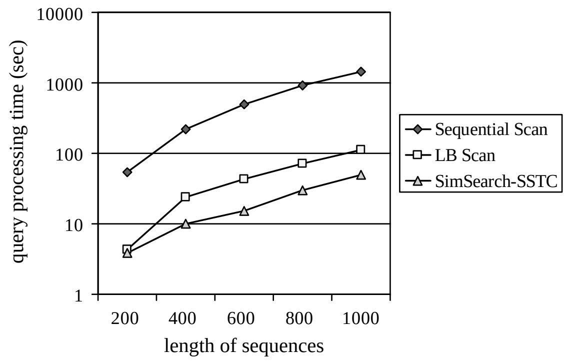


Fig. 12. Average query processing time with the increasing average length of data sequences. The L_1-based time-warping distance was used on the random walk data set.

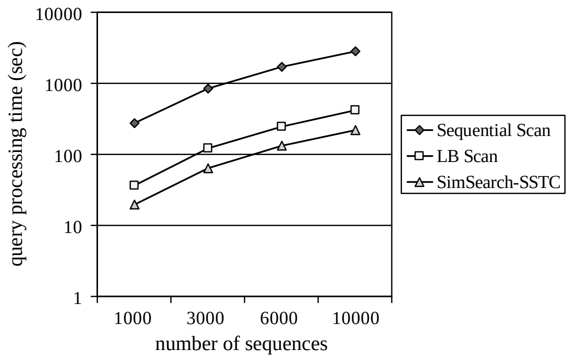


Fig. 13. Average query processing time with the increasing number of data sequences. The L_1-based time-warping distance was used on the random walk data set.

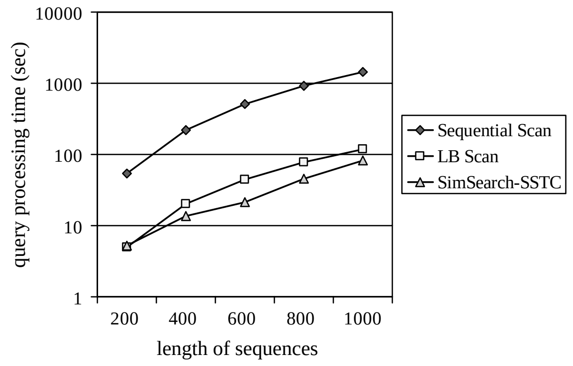


Fig. 14. Average query processing time with the increasing average length of data sequences. The L_2-based time-warping distance was used on the random walk data set.

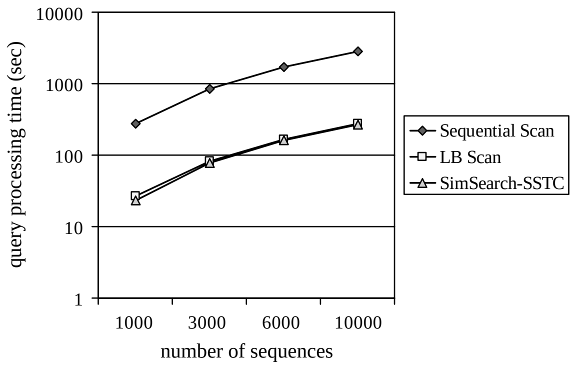


Fig. 15. Average query processing time with the increasing number of data sequences. The L_2-based time-warping distance was used on the random walk data set.

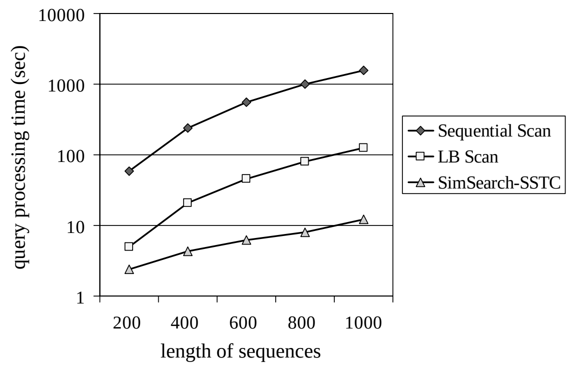


Fig. 16. Average query processing time with the increasing average length of data sequences. The L_∞-based time-warping distance was used on the random-walk data set.

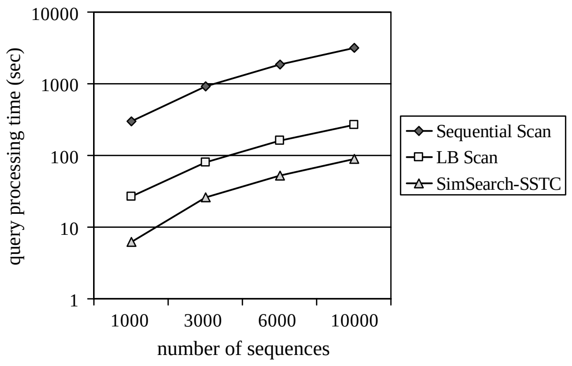


Fig. 17. Average query processing time with the increasing number of data sequences. The L_∞-based time-warping distance was used on the random walk data set.

warping distance metrics, respectively. As seen from these figures, the performance improvement of SimSearch-SSTC is maintained with any value of p in the L_p-based time-warping distance metric even when the database contains a large number of data sequences.

8. Conclusion

This paper presented 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 the time-warping distance as a similarity measure that allows stretching or compressing of sequences along the time axis. Extensive experiments with real and synthetic data sequences revealed that the proposed approach significantly outperforms the sequential and LB scan approaches and scales well in a large volume of sequence databases. The contributions of our work are: (1) extending the exact search algorithm of a suffix tree to the similarity search algorithm with the time-warping similarity measure, (2) applying the idea of categorization and sparse indexing to reduce the index size and to accelerate the query processing, and (3) introducing the search algorithms that use the lower-bound distance functions and the branch-pruning technique to filter out dissimilar subsequences in index space without false dismissal.

The index space can be reduced further if we know the minimum and maximum lengths of the queries. Using a warping window constraint [6], 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 search 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 frequent visiting patterns of web sites.

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) [25]. 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.

References

[1] R. Agrawal, C. Faloutsos, A. Swami, Efficient similarity search in sequence databases, in: Proceedings of International Conference on Foundations of Data Organization and Algorithms (FODO), Chicago, IL, 1993, pp. 69–84.

[2] R. Agrawal, K. Lin, H.S. Sawhney, K. Shim, Fast similarity search in the presence of noise, scaling, and translation in time-series databases, in: Proceedings of the International Conference on Very Large Data Bases (VLDB), Zurich, 1995, pp. 490–501.

[3] C. Faloutsos, M. Ranganathan, Y. Manolopoulos, Fast subsequence matching in time-series databases, in: Proceedings of the ACM International Conference on Management of Data (SIGMOD), Minneapolis, MN, 1994, pp. 419–429.

[4] B.-K. Yi, H.V. Jagadish, C. Faloutsos, Efficient retrieval of similar time sequences under time warping, in: Proceedings of the IEEE International Conference on Data Engineering (ICDE), Orlando, FL, 1998, pp. 201–208.

[5] L. Rabinar, B.-H. Juang, Fundamentals of Speech Recognition, Prentice-Hall, Englewood Cliffs, NJ, 1993.

[6] D.J. Berndt, J. Clifford, Finding patterns in time series: a dynamic programming approach, in: U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, R. Uthurusamy (Eds), Advances in Knowledge Discovery and Data Mining, AAAI/MIT, 1996, Cambridge, MA, pp. 229–248.

[7] D. Rafiei, A. Mendelzon, Similarity-based queries for time series data, in: Proceedings of the ACM International Conference on Management of Data (SIGMOD), Tucson, AZ, 1997, pp. 13–24.

[8] G.A. Stephen, String Searching Algorithms, World Scientific, Singapore, 1994.

[9] N. Beckmann, H. Kriegel, R. Schneider, B. Seeger, The R*-tree: an efficient and robust access method for points and rectangles, in: Proceedings of the ACM SIGMOD, Atlantic City, NJ, 1990, pp. 322–331.

[10] D.Q. Goldin, P.C. Kanellakis, On similarity queries for time-series data: constraint specification and implementation, in: Proceedings of the Constraint Programming, Cassis, France, 1995, pp. 137–153.

[11] A. Guttman, R-trees: a dynamic index structure for spatial searching, in: Proceedings of the ACM SIGMOD, Boston, MA, 1984, pp. 47–57.

[12] T. Bozkaya, N. Yazdani, M. Ozsoyoglu, Matching and indexing sequences of different lengths, in: Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM), Las Vegas, NA, 1997, pp. 128–135.

[13] C. Faloutsos, K. Lin, Fastmap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets, in: Proceedings of the ACM International Conference on Management of Data (SIGMOD), San Jose, CA, 1995, pp. 163–174.

[14] E.J. Keogh, M.J. Pazzani, Scaling up dynamic time warping to massive datasets, in: Proceedings of the Principles and Practice of Knowledge Discovery in Databases (PKDD), Prague, 1999.

[15] E.J. Keogh, M.J. Pazzani, Scaling up dynamic time warping for data mining applications, in: Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Boston, MA, 2000, pp. 285–289.

[16] S. Park, D. Lee, W.W. Chu, Fast retrieval of similar subsequences in long sequence databases, in: Proceedings of the IEEE Knowledge and Data Engineering Exchange Workshop (KDEX), Chicago, IL, 1999, pp. 60–67.

[17] R. Agrawal, G. Psaila, E.L. Wimmers, M. Zait, Querying shapes of histories, in: Proceedings of the International Conference on Very Large Data Bases (VLDB), Zurich, 1995, pp. 502–514.

[18] H. Shatkay, S.B. Zdonik, Approximate queries and representations for large data sequences, in: Proceedings

of the IEEE International Conference on Data Engineering (ICDE), Houston, TX, 1994, pp. 536–545.

[19] P. Bieganski, J. Riedl, J.V. Carlis, Generalized suffix trees for biological sequence data: applications and implementation, in: Proceedings of the Hawaii International Conference on System Sciences, Wailea, Hawaii, 1994.

[20] J.T. Wang, G. Chirn, T.G. Marr, B. Shapiro, D. Shasha, K. Zhang, Combinatorial pattern discovery for scientific data: some preliminary results, in: Proceedings of the ACM International Conference on Management of Data (SIGMOD), Minneapolis, MN, 1994, pp. 115–125.

[21] S. Park, S.W. Kim, J.S. Cho, S. Padmanabhan, Prefix-querying: an approach for effective subsequence matching under time warping in sequence databases, in: Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM), Atlanta, GA, 2001, pp. 255–262.

[22] S. Park, W.W. Chu, J. Yoon, C. Hsu, A suffix tree for fast similarity searches of time-warped subsequences in sequence databases, Technical Report UCLA-CS-TR-990005, UCLA, 1999.

[23] K. Fukunaga, P.M. Narendra, A branch and bound algorithms for computing k-nearest neighbors, IEEE Trans. Comput. C-24 (7) (1975) 750–753.

[24] T. Brinkhoff, H.-P. Kriegel, R. Schneider, B. Seeger, Multi-step processing of spatial joins, in: Proceedings of the ACM International Conference on Management of Data (SIGMOD), Minneapolis, MN, 1994, pp. 237–246.

[25] W.W. Chu, K. Chiang, Abstraction of high level concepts from numerical values in databases, in: Proceedings of the AAAI Workshop on Knowledge Discovery in Databases, Seattle, WA, 1994, pp. 133–144.

[26] C.E. Shannon, W. Weaver, The Mathematical Theory of Communication, University of Illinois Press, Champaign, IL, 1964.

[27] J. Karkkainen, E. Ukkonen, Sparse suffix trees, in: Proceedings of Computing and Combinatorics (COCOON), Hong Kong, 1996, pp. 219–230.