Available online at www.sciencedirect.com

SCIENCE DIRECT®

The Journal of Systems and Software 79 (2006) 191–203

The Journal of Systems and Software

www.elsevier.com/locate/jss

Shape-based retrieval in time-series databases

Sang-Wook Kim [a,*], Jeehee Yoon [b], Sanghyun Park [c], Jung-Im Won [c]

[a] *College of Information and Communications, Hanyang University, 17 Haengdang, Seongdong, Seoul 133-791, Republic of Korea*
[b] *Division of Information Engineering and Telecommunications, Hallym University, 39 Hallymdaehak-gil, Chuncheon, Kangwon 200-702, Republic of Korea*
[c] *Department of Computer Science, Yonsei University, 134 Sinchon, Seodaemoon Gu, Seoul 120-749, Republic of Korea*

Received 24 May 2004; received in revised form 5 May 2005; accepted 7 May 2005
Available online 22 June 2005

Abstract

The *shape-based retrieval* is defined as the operation that searches for the (sub)sequences whose shapes are similar to that of a query sequence regardless of their actual element values. In this paper, we propose a similarity model suitable for shape-based retrieval and present an indexing method for supporting the similarity model. The proposed similarity model enables to retrieve similar shapes accurately by providing the combination of multiple shape-preserving transformations such as normalization, moving average, and time warping. Our indexing method stores every distinct subsequence concisely into the disk-based suffix tree for efficient and adaptive query processing. We allow the user to dynamically choose a similarity model suitable for a given application. More specifically, we allow the user to determine the parameter p of the distance function L_p when submitting a query. The result of extensive experiments revealed that our approach not only successfully finds the subsequences whose shapes are similar to a query shape but also significantly outperforms the sequential scan method.
© 2005 Elsevier Inc. All rights reserved.

Keywords: Similarity search; Shape-based retrieval; Time-series databases

1. Introduction

The time-series database is a set of data sequences (hereafter, we simply call them sequences), each of which is an ordered list of elements (Agrawal et al., 1993). Sequences of stock prices, money exchange rates, temperature data, product sales data, and company growth rates are the typical examples of time-series databases (Agrawal et al., 1995a; Faloutsos et al., 1994). Similarity search is an operation that finds sequences or subsequences whose changing patterns are similar to that of a given query sequence (Agrawal et al., 1993, 1995a; Faloutsos et al., 1994). Similarity search is of growing importance in many new applications such as data mining and data warehousing (Chen et al., 1996; Rafiei and Mendelzon, 1997).

In order to measure the similarity of any two sequences of length n, most approaches (Agrawal et al., 1993; Chu and Wong, 1999; Faloutsos et al., 1994; Goldin and Kanellakis, 1995; Rafiei and Mendelzon, 1997; Rafiei, 1999) map the sequences into points in n-dimensional space and compute the Euclidean distance between those points as a similarity measure. However, they often miss the data sequences that are actually similar to a query sequence in users' perspective. Therefore, recent work on similarity search tends to support various types of transformations such as scaling (Agrawal et al., 1995a; Chu and Wong, 1999), shifting (Agrawal et al., 1995a; Chu and Wong, 1999), normalization (Agrawal et al., 1995a; Chu and Wong, 1999; Das

* Corresponding author. Tel.: +82 2 2220 1736; fax: +82 2 2220 1886.
E-mail addresses: wook@hanyang.ac.kr (S.-W. Kim), jhyoon@hallym.ac.kr (J. Yoon), sanghyun@cs.yonsei.ac.kr (S. Park), jiwon@cs.yonsei.ac.kr (J.-I. Won).

0164-1212/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2005.05.004

et al., 1997; Goldin and Kanellakis, 1995; Loh et al., 2001), moving average (Loh et al., 2000; Rafiei and Mendelzon, 1997; Rafiei, 1999), and time warping (Berndt and Clifford, 1996; Kim et al., 2001; Park et al., 2000, 2001; Yi et al., 1998).

This paper addresses the problem of *shape-based retrieval* that finds the sequences whose shapes are similar to that of a given query sequence regardless of their actual element values. To provide a flexible solution to this problem, this paper introduces a new similarity model that employs combinations of multiple transformations such as shifting, scaling, moving average, and time warping.

In particular, our similarity model supports multiple L_p distance functions in order to measure the similarity between the finally transformed two sequences; If a user chooses one among the Manhattan distance L_1, the Euclidean distance L_2, and the maximum distance L_∞, the proposed method performs the shape-based retrieval by using the chosen distance function. The flexibility in choosing multiple distance functions is fairly useful since users could have different opinions about similarity depending on applications. In addition, an important feature of the proposed method is to perform the shape-based retrieval that supports these three distance functions by using only one index built in advance.

Similarity search is classified into *whole matching* and *subsequence matching* (Agrawal et al., 1993).

- *Whole matching:* Given N data sequences $S_1, \ldots, S_N$, a query sequence Q, and a tolerance ε, we find such data sequences S_i that are similar to Q. Here, we note that the data and query sequences should be of the same length.
- *Subsequence matching:* Given N data sequences $S_1, \ldots, S_N$ of varying lengths, a query sequence Q, and the tolerance ε, we find all the sequences S_i, one or more subsequences of which are similar to Q, and the offsets in S_i of those subsequences. Here, the data and query sequences are allowed to be of arbitrary lengths.

Since subsequence matching is a generalization of whole matching, it is applicable to practical applications more than whole matching.

In this paper, we propose a novel method for processing of *shape-based subsequence retrieval*. We first define an effective similarity model for *shape-based subsequence retrieval* and present the indexing and query processing methods for performing the shape-based retrieval that supports this model. To verify the superiority of the approach, we perform extensive experiments by using a variety of data sets. The results reveal that our approach successfully finds all the subsequences that have the shapes similar to that of the query sequence, and also achieves high search performance.

This paper is organized as follows. Section 2 briefly reviews previous work related to similarity search. Section 3 defines the notation and terminology used in this paper and introduces our similarity model. Section 4 presents the indexing method for supporting the proposed similarity model, and Section 5 describes our query processing method. Section 6 presents the experimental results to show the superiority of our method, and finally, Section 7 summarizes and concludes the paper.

2. Related work

In this section, we briefly survey previous research results associated with similarity search in time-series databases.

Agrawal et al. (1993) proposed a method for whole matching in time-series databases. First, each data sequence of length n is transformed into a point in f ($\ll n$) dimensional space by using the *discrete Fourier transform* (DFT). For indexing a large number of such points, an R^*-tree (Beckmann et al., 1990) is used. For whole matching, a query sequence of length l is also transformed into a point over f-dimensional space in the same way, and the R^*-tree is traversed to perform the range query by using the transformed query point. As a result, candidate sequences, which are highly likely to be the final answers, are found. Then, every candidate sequence is accessed from disk, and its actual distance to the query sequence is computed. If the distance is smaller than ε, the candidate sequence is returned as a final answer.

Faloutsos et al. (1994) and Moon et al. (2001) proposed methods for processing of subsequence matching. They use the concept of a *window*, which is a fixed-sized subsequence inside the query and data sequences. Each window of length w is transformed into a window point in f ($\ll w$) dimensional space by using the DFT or the wavelet transform. Such window points are indexed by an R^*-tree (Beckmann et al., 1990). For subsequence matching, windows are extracted from a query sequence and are transformed into window points in f-dimensional space. For each window point, a range query is performed on the R^*-tree to obtain candidate subsequences, each of which has a high possibility to be included in the final result. Finally, each candidate subsequence is accessed from disk, and its actual Euclidean distance to the query sequence is examined.

In some applications, similarity search based only on Euclidean distance often fails to search for the data sequences that are actually similar to a query sequence in users' perspective. Therefore, to give flexibility on defining of the similarity, recent work tends to support various types of transformations.

Agrawal et al. (1995a), Chu and Wong (1999), Das et al. (1997), Goldin and Kanellakis (1995), Loh et al. (2000) and Rafiei (1999) proposed the methods for similarity search that supports *normalization*. Normalization enables finding sequences that have a fluctuation pattern similar to that of a query sequence even through they are not close to each other before the normalization. Das et al. (1997) suggested a whole matching method that finds similar sequences by accessing an entire database. Also, Agrawal et al. (1995a) and Goldin and Kanellakis (1995) proposed methods that exploit R^*-trees as indexes to improve the search performance significantly. While Agrawal et al. (1995a), Das et al. (1997) and Goldin and Kanellakis (1995) dealt with only whole matching, Chu and Wong (1999) and Loh et al. (2000) extended their idea to process subsequence matching effectively by using indexes.

Loh et al. (2001), Rafiei and Mendelzon (1997) and Rafiei (1999) dealt with the methods for similarity search that supports *moving average transformation*. The moving average transformation converts a given data sequence into a new sequence consisting of the averages of k consecutive values in the data sequence, where k is called the moving average coefficient. The moving average transformation is very useful for finding the trend of the time-series data by reducing the effect of noise inside. Rafiei and Mendelzon (1997) proposed a whole matching method that employs the convolution definition (Preparata and Shamos, 1985) to support moving average transformation of arbitrary coefficient using only one index. Loh et al. (2000) pointed out the problems in applying the previous methods to subsequence matching, and suggested a method that performs subsequence matching effectively by using the concept of *index interpolation*.

Berndt and Clifford (1996), Kim et al. (2001), Park et al. (2000, 2001), Rafiei and Mendelzon (1997), and Yi et al. (1998) addressed the issue of supporting *time warping* within the similarity model. Time warping is a transformation that allows any sequence element to replicate itself as many times as needed, and is useful in situations where sequences are of varying lengths and so their similarity cannot be directly computed by applying the L_p distance function. Berndt and Clifford (1996) and Yi et al. (1998) proposed whole matching methods, which access and examine every sequence in databases. Kim et al. (2001), Park et al. (2000), and Yi et al. (1998) discussed approaches that exploit indexes to perform whole matching efficiently. Also, Park et al. (2001) extended the basic idea of Kim et al. (2001) to process subsequence matching effectively by using the concept of *prefix-querying*.

As stated earlier, most previous approaches support only one transformation in their similarity model. In time-series database applications, however, it is frequently required to retrieve data (sub)sequences whose shapes are similar to that of a query sequence regardless of their actual element values. In this paper, we define this kind of a problem as the shape-based retrieval.

Table 1
Notation

Notation	Description
$S = (s[i])$	Data sequence ($0 \leqslant i < \text{Len}(S)$)
Len(S)	Number of elements in S
$X = (x[i])$	Arbitrary subsequence of S ($0 \leqslant i < \text{Len}(X) \leqslant \text{Len}(S)$)
$Q = (q[i])$	Query sequence ($0 \leqslant i < \text{Len}(Q)$)
ε	Distance tolerance
First(S)	The first element of S, $s[0]$
Rest(S)	Sequence composed of all the elements of S except for the first element, $(s[1], s[2], \ldots, s[\text{Len}(S) - 1])$
Max(S)	The largest element of S
Min(S)	The smallest element of S
()	Empty sequence
Max(a, b)	The largest value between a and b
Min(a, b, c)	The smallest value among a, b, and c

Agrawal et al. (1995b) allowed a user to specify a required query pattern using their *shape definition language* (SDL). Given a query pattern defined by SDL, they utilized the hierarchical index structure to retrieve the subsequences satisfying the query pattern. This method converts numeric elements into their corresponding symbols and compares the symbol sequences without considering their original element values. Two numeric elements whose values are very close to each other could be converted into different symbols in this method. Therefore, this method may judge quite similar sequences to be dissimilar.[1]

Perng et al. (2000) proposed the *landmark model* for shape-based pattern matching. The landmark model extracts the landmarks from the sequences, and compares any two sequences using their landmarks without examining their original element values. The landmark model enables to find the sequences of similar shapes intuitively. However, the landmarks are highly dependent on target applications, thus making it complicated to find the landmarks suitable for a specific application.

3. Problem definition

This section formally defines the problem we are going to solve. Section 3.1 defines the notation and terminology used in this paper. Section 3.2 describes our similarity model.

3.1. Notation and terminology

Table 1 defines the notation frequently used in this paper. A *data sequence* is a sequence stored in a database

[1] By the categorization stated in Section 4, our method would also convert similar values into different symbols. Unlike the method in Agrawal et al. (1995b) that only compares two symbolized sequences without considering their values, our method makes value-based comparisons in index traversal by using a lower-bound distance function $D_{\text{tw-lb}}$. Thus, our method does not make such a wrong judgment due to the symbolization.

and a *query sequence* is a sequence submitted for querying.

Definition 1. Given a sequence $S = (s[i])$ $(0 \leqslant i < \mathrm{Len}(S))$, its normalized sequence $\mathrm{Norm}(S) = (s'[i])$ $(0 \leqslant i < \mathrm{Len}(S))$ is defined as follows (Agrawal et al., 1995a)[2]:

$$s'[i] = \frac{s[i] - \frac{\mathrm{Max}(S)+\mathrm{Min}(S)}{2}}{\frac{\mathrm{Max}(S)-\mathrm{Min}(S)}{2}}.$$

Normalization, a combination of shifting and scaling, is a transformation that reduces the effect of absolute element values. Therefore, it is useful in finding the sequences with similar changing patterns even though their absolute values may be different. For example, consider two sequences S_1 and S_2 in Fig. 1. Although S_1 and S_2 have different element values, we see that their changing patterns are quite similar. By normalization, they are transformed into identical sequences $\mathrm{Norm}(S_1)$ and $\mathrm{Norm}(S_2)$.

Definition 2. Given a sequence $S = (s[i])$ $(0 \leqslant i < \mathrm{Len}(S))$ and a moving average coefficient k, S is transformed into $\mathrm{MV}_k(S) = (s_k[j])$ $(0 \leqslant j < \mathrm{Len}(S) - k + 1)$ by k-moving average transformation (Chatfield, 1984; Kendall, 1979):

$$s_k[j] = \frac{s[j] + s[j+1] + \cdots + s[j+k-1]}{k} = \frac{\sum_{l=j}^{j+k-1} s[l]}{k}.$$

As described in Definition 2, the k-moving average transformation generates a series of elements from the average values of successive k elements of an original sequence. The moving average transformation reduces the effect of noises embedded in sequences. Therefore, it is useful to detect sequences of similar changing patterns without worrying about noises. Users decide the moving average coefficient k according to the intention of how much they want to reduce the effect of noises. For example, Fig. 2 shows sequence S, and its 4- and 8-moving averaged sequences $\mathrm{MV}_4(S)$ and $\mathrm{MV}_8(S)$. We see that the effect of noises reduces as the moving average coefficient k increases.

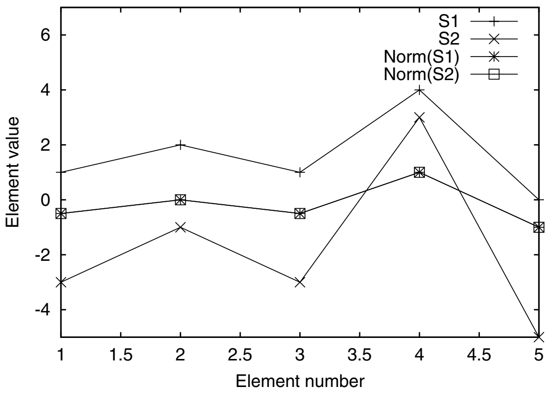

Fig. 1. Example of normalization transformation.

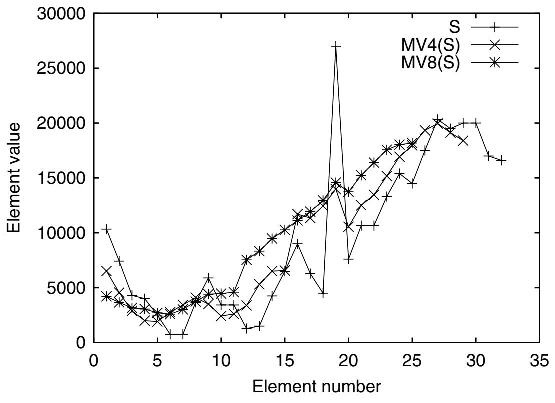

Fig. 2. Example of moving average transformation.

Definition 3. Given two sequences S and Q of the same length n, the distance function L_p is defined as follows. L_1 is the Manhattan distance, L_2 is the Euclidean distance, and L_∞ is the maximum distance in any pair of elements (Shim et al., 1997).

$$L_p(S,Q) = \left(\sum_{i=1}^{n} |s[i] - q[i]|^p\right)^{1/p}, \quad 1 \leqslant p \leqslant \infty.$$

Even though the distance function L_p is widely used in many applications, it has the strict restriction that the two sequences to be compared should be of the same length. Time warping is a transformation that allows any sequence element to replicate itself as many times as needed without extra costs (Yi et al., 1998). For example, two sequences $S = (20, 21, 21, 20, 20, 23, 23, 23)$ and $Q = (20, 20, 21, 20, 23)$ can be identically transformed into $(20, 20, 21, 21, 20, 20, 23, 23, 23)$ by time warping. The *time warping distance* is defined as the smallest distance between two sequences transformed by time warping.

Definition 4. Given two sequences S and Q, the time warping distance D_{tw} is defined recursively as follows

[2] We may use a different type of normalization (Goldin and Kanellakis, 1995; Rafiei and Mendelzon, 1997) that first computes the average $\mathrm{avg}(S)$ and the standard deviation $\mathrm{std}(S)$ from S, and then replaces each element $s[i]$ with $s'[i] = \frac{s[i]-\mathrm{avg}(S)}{\mathrm{std}(S)}$. However, we employ the normalization defined in Definition 1 in order to make all elements take the values within the range $[-1.0, 1.0]$. This enables high compression of a *subsequence tree*, which is defined in Section 4. Compared with using of $\mathrm{avg}(S)$ and $\mathrm{std}(S)$, the normalization defined in Definition 1 tends to be sensitive to noises embedded in sequences. As mentioned in Section 3.2, however, our similarity model first eliminates such noises in each sequence through moving average transformation before doing normalization transformation. Thus, we can safely employ $\mathrm{Min}(S)$ and $\mathrm{Max}(S)$ instead of $\mathrm{avg}(S)$ and $\mathrm{std}(S)$ without worrying about the noise effect.

(Rabiner and Juang, 1993). Here, D_{base} can be any L_p function that returns the distance of two elements.

$$\begin{aligned}
&D_{\text{tw}}((),()) = 0,\\
&D_{\text{tw}}(S,()) = D_{\text{tw}}((),Q) = \infty,\\
&D_{\text{tw}}(S,Q) = \big((D_{\text{base}}(\text{First}(S),\text{First}(Q)))^p\\
&\qquad + (\text{Min}(D_{\text{tw}}(S,\text{Rest}(Q)), D_{\text{tw}}(\text{Rest}(S),Q),\\
&\qquad D_{\text{tw}}(\text{Rest}(S),\text{Rest}(Q))))^p\big)^{1/p}.
\end{aligned}$$

While the Euclidean distance can be used only when two sequences compared are of the same length, the time warping distance can be applied to any two sequences of arbitrary lengths. Therefore, the time warping distance is smoothly applicable to the databases where sequences are of different lengths.

3.2. Similarity model

The goal of this work is to devise a method that effectively finds the subsequences whose shapes are similar to that of a query sequence. To support the shape-based retrieval, this paper defines the following similarity model that combines shifting, scaling, moving average, and time warping transformations together.

Definition 5. Given two sequences or subsequences S and Q, their distance (or dissimilarity) is defined as follows:

$$D(S,Q) = D_{\text{tw}}(\text{Norm}(\text{MV}_k(S)), \text{Norm}(\text{MV}_k(Q))).$$

As in Definition 5, the distance of S and Q is defined as the time warping distance of two sequences converted by (1) k-moving average transformation, and then (2) normalization transformation.

The shape-based retrieval based on our similarity model supports shifting, scaling, and time warping transforms, and also minimizes the effect of noises owing to moving average transformation. In particular, when computing the time warping distance between the two transformed sequences, we provide three types of L_p (the Manhattan distance L_1, the Euclidean distance L_2, and the maximum distance L_∞) as a base distance function. These distance functions have the following features (Agrawal et al., 1993; Sidiropoulos and Bros, 1999; Yi and Faloutsos, 2000).

It has been known that L_1 is optimal when errors are additive, i.i.d. (independent, identically distributed) Laplacian (or double exponential). Thus, it is more robust against impulsive noise. L_2 is optimal in the maximum likelihood sense when errors are additive, i.i.d. Gaussian. It has been the most popular dissimilarity measure in time-series applications. L_∞ has an advantage that users can easily specify a tolerance without considering the length of a query sequence. It has a disadvantage of being sensitive to noises within a sequence. As stated earlier, however, our similarity model first performs the moving average transformation, thereby removing the noises in each sequence. Therefore, we do not need to worry about the noise effect in applying L_∞.

Applications require different target query results according to their characteristics. Also, users even in the same application tend to want different target query results according to their propensities. In this work, we propose a similarity model that supports all of L_1, L_2, and L_∞ as a base distance function in order to give users choices at querying time. Users can choose their own distance function according to their preferences, and thus, get what they want to retrieve from a database. In Section 6, we present experimental results by different L_p distance functions.

We define the problem of the shape-based retrieval in a time-series database as follows: Given a query sequence Q, a distance tolerance ε, a moving average coefficient k and p for the L_p distance function, we find the subsequences X whose distances to Q $(D(X,Q) = (D_{\text{tw}}(\text{Norm}(\text{MV}_k(X)), \text{Norm}(\text{MV}_k(Q)))))$ are smaller than ε. As a result, we get the sequences S that contain X and the starting offset of X inside S.

4. Indexing

This section describes the indexing method for efficient shape-based retrieval. Section 4.1 briefly reviews the suffix tree, the underlying index structure in our method. Section 4.2 discusses the indexing strategy for utilizing the suffix tree. Section 4.3 describes the index construction steps in detail, and Section 4.4 presents the technique for index compression.

4.1. Suffix tree

A *trie* is a data structure for indexing a set of keywords of varying sizes. A *suffix tree* (Stephen, 1994) 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* (Stephen, 1994). A suffix tree is generalized to allow multiple sequences to be stored in the same tree, and is useful to find the subsequences exactly matched with a query sequence. A suffix tree does not assume any distance function for its construction. Therefore, it can support various distance functions at query processing time.

Each suffix of a sequence is represented by a leaf node. For example, given sequence $S_i = (s[0], s[1], \ldots, s[\text{Len}(S_i) - 1])$, its suffix $(s[j], s[j+1], \ldots, s[\text{Len}(S_i) - 1])$ is represented by the leaf node labeled with (S_i,j). The edges are labeled with the subsequences such that

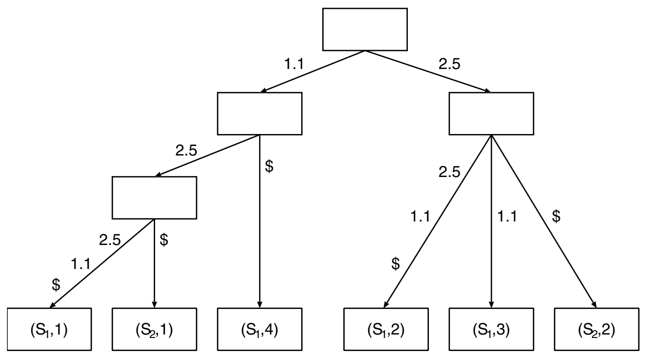

Fig. 3. Suffix tree from $S_1 = (1.1, 2.5, 2.5, 1.1)$ and $S_2 = (1.1, 2.5)$.

the concatenation of the edge labels on the path from the root to the leaf labeled with (S_i, j) becomes suffix $(s[j], s[j+1], \ldots, s[\mathrm{Len}(S_i) - 1])$. The concatenation of the edge labels on the path from the root to internal node N represents the longest common prefix of the suffixes represented by the leaf nodes under N. Fig. 3 shows the suffix tree constructed from two sequences $S_1 = (1.1, 2.5, 2.5, 1.1)$ and $S_2 = (1.1, 2.5)$. Four suffixes ((1.1, 2.5, 2.5, 1.1), (2.5, 2.5, 1.1), (2.5, 1.1), and (1.1)) from S_1 and two suffixes ((1.1, 2.5) and (2.5)) from S_2 are extracted and then inserted into the suffix tree. The symbol $ is used as the end marker of a suffix.

The suffix tree becomes more compact when the suffixes contain more and longer common prefixes. However, it is rare for the suffixes to have common prefixes because every element takes values from a continuous domain. Park et al. (2000) proposed to use *categorization* to solve this problem. Categorization divides the entire range from which elements take their values into multiple non-overlapping subranges. Then, every element is converted into the symbol of its corresponding subrange. If we build the suffix tree from sequences of symbols, the suffixes are likely to have the common prefixes much more than before.

4.2. Indexing strategy

Consider a suffix S_a of a sequence S and a prefix S_b of S_a. Since S_b is the prefix of S_a, every element of S_b is contained in S_a. Therefore, we can obtain the distance of S_b and Q while we compute the time warping distance between S_a and Q. Thus, if we do not consider normalization, the index constructed from the suffixes is enough for retrieving any subsequences.

However, our similarity model supports normalization. As indicated in Definition 1, we use $\mathrm{Max}(S)$ and $\mathrm{Min}(S)$ to normalize (sub)sequence S. Let us consider the suffix S_a and its prefix S_b again. Since $\mathrm{Max}(S_a)$ and $\mathrm{Min}(S_a)$ can be different from $\mathrm{Max}(S_b)$ and $\mathrm{Min}(S_b)$, we can not guarantee that $\mathrm{Norm}(S_b)$ is the prefix of $\mathrm{Norm}(S_a)$. Therefore, the distance of $\mathrm{Norm}(S_b)$ and Q is no longer a by-product of the distance computation between $\mathrm{Norm}(S_a)$ and Q. As a result, we have to store every possible subsequence in the tree in order to support our similarity model safely. In this paper, we call the tree, which has the same structure as the suffix tree but stores every possible subsequence, the *subsequence tree*.

While the suffix tree stores n suffixes from a sequence with n elements, the subsequence tree stores $\frac{n(n+1)}{2}$ subsequences. However, the subsequence tree does not grow excessively because the subsequences from the same sequence have the high possibility to share common prefixes.

4.3. Construction of subsequence tree

To build a subsequence tree from a time-series database, we take the following five steps.

- *Step 1: Moving average transformation*
  After selecting the moving average coefficient k suitable for a given target application, we perform k-moving average transformation for *every sequence* stored in a database. $\mathrm{MV}_k(S)$ denotes the sequence converted from S by k-moving average transformation.
- *Step 2: Subsequence extraction*
  We extract every possible subsequence X from $\mathrm{MV}_k(S)$. Note that X is not a subsequence of S but a subsequence of $\mathrm{MV}_k(S)$. At this step, we can determine the minimum length L of subsequences to be extracted. That is, when it is meaningless to retrieve too short subsequences as answers, we extract only the subsequences whose lengths are longer than or equal to L. This contributes towards reducing the size of the subsequence tree.
- *Step 3: Normalization transformation*
  We normalize every subsequence X using $\mathrm{Max}(X)$ and $\mathrm{Min}(X)$. $\mathrm{Norm}(X)$ denotes the normalized sequence of X.
- *Step 4: Symbolization using categorization*
  We first decide a set of categories (subranges) by examining the distribution of element values in a database after step 3, and assign a unique symbol to each category. Then, we convert each element of $\mathrm{Norm}(X)$ into the symbol of the corresponding category. The number of categories depends on a target application (Park et al., 2000).
- *Step 5: Tree construction*
  We build a subsequence tree from a set of symbolized subsequences. Each symbolized subsequence is represented by the path from the root to a leaf node. The identifier (SID, i, j) of a symbolized subsequence is stored in the corresponding leaf node where SID is the identifier of a sequence from which the symbolized subsequence has been obtained, i is its starting

offset, and j is its ending offset. Since the subsequence tree is possibly constructed from a large volume of a database, we employ the disk-based algorithm (Park et al., 2000) that reduces the number of disk accesses.

4.4. Index compression

The subsequence tree does not grow excessively even when the database becomes large. This is because the subsequence tree has a large potential to compress the input subsequences. However, it is still true that the subsequence tree from fewer subsequences is smaller than the one from more subsequences. This subsection presents the technique, which reduces the number of subsequences to be stored in the subsequence tree without losing any necessary information for query processing.

Let us consider two subsequences S_a and S_b where S_b is a prefix of S_a. Norm(S_b) is still prefix of Norm(S_a) when $\text{Max}(S_a) = \text{Max}(S_b)$ and $\text{Min}(S_a) = \text{Min}(S_b)$. Therefore, given a query sequence Q, the distance between Norm(S_b) and Q can be obtained as a by-product of the distance computation between Norm(S_a) and Q. This implies that S_b does not have to be inserted into the subsequence tree as long as S_a is stored in the tree. Let $S[i{:}j]$ denote the subsequence of S including elements in positions i through j. The procedure to determine whether $S[i{:}j]$ is to be inserted into the tree or not is formally defined as follows: (1) insert $S[i{:}j]$ if $j = \text{Len}(S) - 1$ (that is, there is no subsequence which contains $S[i{:}j]$ as a prefix), (2) insert $S[i{:}j]$ if $\text{Max}(S[i{:}j]) \neq \text{Max}(S[i{:}j+1])$ or $\text{Min}(S[i{:}j]) \neq \text{Min}(S[i{:}j+1])$. We call the tree, which stores the set of those subsequences which pass the above test procedure, the *compact subsequence tree*.

5. Query processing

This section presents the query processing method for shape-based retrieval of similar subsequences, and shows its computation complexity.

5.1. Algorithm

We premise that users submit a noise-free query sequence by using either one of two ways. The first way is to make users directly determine the element values of the query sequence. The query sequence thus obtained is free from the noises since users draw the specific shape of a sequence that they want to find. The second way is to make users select a subsequence from a sequence chosen in a database, and also transform it into a k-moving averaged subsequence. In both ways, we are free from the noise effect, and thus safely regard both query sequences to be k-moving averaged. As a next step, we normalize the query sequence, and then perform the following query processing algorithm using this query sequence Q.

Algorithm 1. Similarity search algorithm using compact subsequence tree Search-CST.

```
Input: compact subsequence tree CST, query sequence
       $Q$, tolerance $\varepsilon$, base distance function $L_p$
Output: set of answers answerSet
1 candidateSet := VisitNode-and-FindAnswers-CST
  (rootNode (CST), $Q$, $\varepsilon$, emptyTable, $L_p$);
2 answerSet := PostProcess (candidateSet);
3 return answerSet;
```

Algorithm 1 shows Search-CST that traverses the compact subsequence tree to retrieve the subsequences whose time warping distances D_{tw} from query sequence Q are within the distance tolerance ε. Remember that we allow users to specify the desired L_p distance function at querying time. Therefore, L_p is given to Algorithm 1 as one of the arguments. We note that the query sequence and the subsequences in the tree have been transformed by k-moving average and normalization.

As described in Section 4.3, a compact subsequence tree is constructed from symbol sequences. Therefore, it is not possible to compute the exact time warping distance between the query sequence and the subsequence already converted into the symbol sequence. Therefore, Search-CST uses the distance function $D_{\text{tw-lb}}$ defined in Park et al. (2000) to compute the lower bound distance between the query sequence and symbolized sequence CS.

Definition 6. Given query sequence Q and symbolized sequence CS, the lower-bound time warping distance function $D_{\text{tw-lb}}(\text{CS}, Q)$ is defined as follows:

$$\begin{aligned}
D_{\text{tw-lb}}((),()) &= 0,\\
D_{\text{tw-lb}}(\text{CS},()) &= D_{\text{tw-lb}}((),Q) = \infty,\\
D_{\text{tw-lb}}(\text{CS},Q) &= \big((D_{\text{base-lb}}(\text{First}(\text{CS}),\text{First}(Q)))^p\\
&\quad + (\text{Min}(D_{\text{tw-lb}}(\text{CS},\text{Rest}(Q)),\\
&\quad D_{\text{tw-lb}}(\text{Rest}(\text{CS}),Q),\\
&\quad D_{\text{tw-lb}}(\text{Rest}(\text{CS}),\text{Rest}(Q))))^p\big)^{1/p}
\end{aligned}$$

$$\begin{aligned}
D_{\text{base-lb}}(A,b) &= 0 && (\text{if } A.\text{lb} \leqslant b \leqslant A.\text{ub})\\
&= b - A.\text{ub} && (\text{if } b > A.\text{ub})\\
&= A.\text{lb} - b && (\text{if } b < A.\text{lb}).
\end{aligned}$$

Here, A is the symbol of First(CS) and b is the actual numeric value of First(Q). A.lb and A.ub denote the minimum and maximum element values of the subrange corresponding to the symbol A.

Algorithm Search-CST calls function VisitNode-and-FindAnswers-CST (line 1) to retrieve the candidate subsequences whose lower bound time warping distances from Q are within ε. Let us consider function VisitNode-and-FindAnswers-CST shown in Algorithm 2. When the algorithm visits node N, it inspects each child node to find new candidates and determines if it is necessary to go further down the tree.

For example, suppose that the algorithm stays on node N and inspects its child node CN_i. For simpler explanation, we assume that each edge is labeled by a single symbol. The algorithm builds the cumulative distance table between Q and label(N, CN_i), locating Q and label(N, CN_i) on the X-axis and on the Y-axis, respectively. If N is the root node, the table is built from the bottom. Otherwise, the table is constructed by augmenting a new row on the table that has been accumulated from the root to N. Function AddRow (line 3) adds a new row using the distance function $D_{\text{tw-lb}}$ with a given L_p as a base distance function. The first step is to inspect the last column of the newly added row to find candidates (line 4). If the last column has a value not larger than ε (line 5), then the algorithm extracts the identifiers from the leaf nodes under CN_i and adds them to the candidate set (line 6).

Note that there may be some data subsequences not stored in a compact subsequence tree. These subsequences are embedded in the paths from the root to *internal* nodes. Therefore, the paths to internal nodes can generate candidate answers. Suppose that the path from the root to the node CN_i has a lower-bound distance not larger than ε from Q. Then, all the subsequences represented by this path can be easily identified by extracting the leaf nodes under CN_i. Let LN be one of such leaf nodes. When LN has an identifier (S_k,j,j') and the length of the path from the root to CN_i is l, one of the subsequences identified by this path is by $(S_k, j, j+l-1)$.

The next step is to determine if it is necessary to go further down the tree. That is, if at least one column of the newly added row has a value not greater than ε (line 8), the search continues to go down the tree to find more candidates. Otherwise, the search moves to the next child of N. When it is necessary to go further down the tree, the algorithm calls itself recursively (line 9).

Since VisitNode-and-FindAnswers-CST finds candidates using the lower-bound distance function $D_{\text{tw-lb}}$, the subsequences whose actual distances are larger than ε may be contained in the candidate set. Those subsequences are called *false alarms* (Faloutsos et al., 1994). Therefore, Search-CST requires the post-processing step to detect and discard false alarms. For each answer in the candidate set, function PostProcess (line 2) retrieves the corresponding subsequence, transforms it using k-moving average and normalization, and then computes its actual distance from Q using the original time warping distance function D_{tw}. The subsequences whose actual time warping distances from Q are not larger than ε are returned to a user as final answers (line 3).

Algorithm 2. Algorithm for traversing the compact subsequence tree VisitNode-and-FindAnswers-CST.

```
Input:  node $N$, query sequence $Q$, tolerance $\varepsilon$,
        cumulative distance table $T$, base distance
        function $L_p$
Output: set of candidate answers candidateSet
1  candidateSet := {};
2  for (each child node $\text{CN}_i$ of the node $N$) do
3      $\text{CT}_i$ := AddRow ($T$, $Q$, label($N$,$\text{CN}_i$), $\text{D}_{\text{tw-lb}}$, $L_p$);
4      Let dist be the last column value of the new row;
5      if (dist $\leqslant \varepsilon$) then
6          candidateSet := candidateSet ∪
           {GetID(GetLeafNodes($\text{CN}_i$))};
7      Let mDist be the minimum column value of the
       new row;
8      if (mDist $\leqslant \varepsilon$) then
9          candidateSet := candidateSet ∪
           VisitNode-and-FindAnswers-CST
           ($\text{CN}_i$, $Q$, $\varepsilon$, $\text{CT}_i$, $L_p$);
10 return candidateSet;
```

5.2. Algorithm analysis

Before analyzing the complexity of Search-CST, let us examine the complexities of Seq-Scan and Search-ST. Seq-Scan is the sequential scan method and Search-ST is the method using a subsequence tree as an index.

Seq-Scan reads each data sequence S, performs k-moving average transformation for S to get $\text{MV}_k(S)$, extracts every possible subsequence X from $\text{MV}_k(S)$, and normalizes every X. Given query sequence Q and subsequence X, it builds the cumulative distance table with the computation complexity of $O(|Q||X|)$ (Berndt and Clifford, 1996). For M k-moving averaged sequences whose average length is L, there are $\frac{ML(L+1)}{2}$ subsequences and their average length is $\frac{L+2}{3}$. Therefore, the computation complexity of Seq-Scan is $O(ML^3|Q|)$.

Search-ST is computationally much cheaper than Seq-Scan due to branch-pruning and sharing of common cumulative distance tables for all the subsequences that have common prefixes. Thus, the complexity of Search-ST is $O\left(\frac{ML^3|Q|}{R_d R_p} + nL|Q|\right)$. The left term is for tree traversal and the right term is for post-processing. Here, R_d ($\geqslant 1$) is the reduction factor due to sharing of the cumulative distance tables, and R_p ($\geqslant 1$) is the reduction factor gained from the branch-pruning. n represents the number of candidate subsequences that require the post-processing.

The complexity of Search-CST is easily derived from that of Search-ST. The compact subsequence tree can be traversed faster than the corresponding subsequence tree because the former is usually smaller than the latter. However, the subsequence tree has a larger reduction factor R_d due to a higher possibility to share common prefixes. Let ρ $(0 < \rho \leqslant 1)$ be the compression ratio of the compact subsequence tree, $\rho =$ the number of stored subsequences/the number of total subsequences. Then the complexity of Search-CST is $O\left(\frac{\rho ML^3|Q|}{R'_d R_p} + nL|Q|\right)$.

6. Performance evaluation

This section presents the experimental results for performance evaluation of the proposed method. Section 6.1 describes the environment for experiments and Section 6.2 shows and analyzes experimental results.

6.1. Environment

All the parameter values for our experiments were selected for simulating the behavior of real-world stock prices and query patterns issued for investment. Two kinds of data sets were used for experiments: synthetic data set and real world stock data set. Each synthetic data sequence $S = \langle s_1, s_2, \ldots, s_n \rangle$ was generated by the following random walk expression:

$$s_i = s_{i-1} + z_i.$$

Here, z_i is an independent, identically distributed (ID) random variable that takes values in the range $[-0.1, 0.1]$. The value of the first element s_1 was taken randomly within the range $[1, 10]$. Stock data sequences were extracted from USA S&P 500 and their element values were based on daily closing prices. As mentioned in Section 5, query sequences were randomly selected from those obtained by k-moving averaging such data sequences.

We have conducted performance evaluation on the three different approaches: Search-CST, Search-ST, and Seq-Scan. Search-CST represents our approach that uses the compact subsequence tree and utilizes a single index structure for all L_p $(p = 1, 2, \infty)$ distance function. Search-CST-L$_1$, Search-CST-L$_2$, and Search-CST-L$_\infty$ represent our approaches that employ L_1, L_2, and L_∞ distance functions, respectively. Search-ST also represents our approach that uses just the subsequence tree. Search-ST-L$_1$, Search-ST-L$_2$, and Search-ST-L$_\infty$ represent Search-ST methods that employ L_1, L_2, and L_∞, respectively. Finally, Seq-Scan is a naive sequential scan method and Seq-Scan-L$_1$, Seq-Scan-L$_2$ and Seq-Scan-L$_\infty$ represent Seq-Scan methods that employ L_1, L_2, and L_∞, respectively.

The hardware platform for the experiments is the Sun UltraSparc-10 workstation equipped with 512 MB RAM. The software platform is Solaris 7, which was set to a single-user mode to minimize the interference from other system and user processes.

6.2. Results and analyses

In Experiment 1, we evaluated the effectiveness and performance of our similarity model with S&P 500 stock data set. We selected 1000 sequences of average length 100 from the data set and applied the 10-moving average transformation. As a query sequence, we used a subsequence of double bottom pattern, which is frequently used in stock data analyses. The values of distance tolerance ε were determined to retrieve 100, 200, 300 and 1000 different answers, respectively.

Fig. 4 shows some instances of the double bottom pattern retrieved by our similarity method. In each figure, the dotted line represents an actual sequence in the data set, and the plain line represents its 10-moving averaged sequence. Fig. 4(a) represents the sequence selected for querying. We extracted a double bottom pattern from that sequence and used it as a query sequence. The remaining figures represent the answers obtained from the data set. The subsequences represented by bold lines in Fig. 4(b)–(d) are some answers retrieved by our methods that employ L_1, L_2, and L_∞, respectively.

We see that all the answers contain the double bottom pattern whose shape is quite similar to that of the query sequence even though their actual element values are multifarious. And these answers illustrate the characteristics of different distance functions well. We believe that the choice of appropriate distance functions for similar subsequence matching is highly application dependent and up to application engineers.

As the proposed similarity model supports moving average and time warping transformations, the answers retrieved by our methods using L_1, L_2, and L_∞ distance functions have many common subsequences. For example, Fig. 4(e) depicts common answers retrieved by our three methods employing L_1, L_2, and L_∞ distance functions. Fig. 5 shows the results of an experiment that compares the answers retrieved by our methods. The stock data sequences were used as a data set and the distance tolerance ε was determined to retrieve 100 different answers. The *Venn Diagram* represents the percentages of common answers retrieved by two or three methods. The experimental results with more answers were similar to those in Fig. 5.

In Experiment 2, we compared the time and space efficiency of the proposed approaches. We selected 200, 400, 600, 800, and 1000 sequences of average length 100 from the stock data set. To construct the index, we set the number of categories (C) to 60. As C increases, the index size also increases but the query processing time decreases. However, when C exceeds a certain threshold value, the index size and the query processing

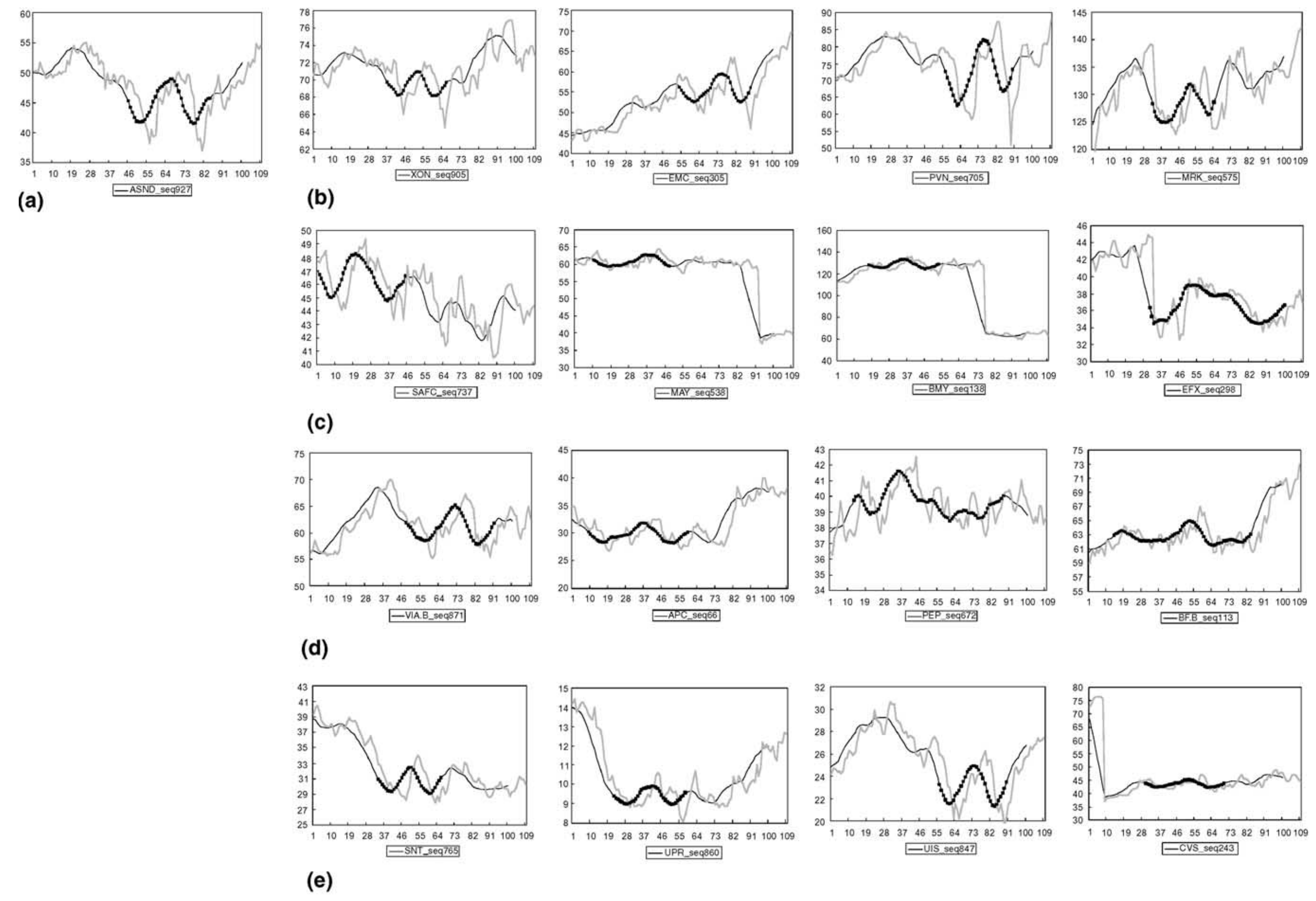

Fig. 4. Example of shape-based retrieval of similar subsequences: (a) Query sequnce, (b) L_1-based, (c) L_2-based, (d) L_∞-based, and (e) L_1, L_2, L_∞-based.

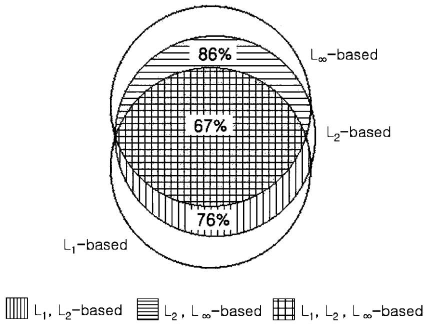
86%
L_∞-based
67%
L_2-based
76%
L_1-based
▥ L_1, L_2-based ▤ L_2, L_∞-based ▦ L_1, L_2, L_∞-based

Fig. 5. Percentages of common answers retrieved by our methods using L_1, L_2, and L_∞ distance functions.

time become nearly constant. We decide this threshold value as the optimal number of categories. In our experiment, we chose 60 as the optimal number of categories, and thus set the number of categories for all the following experiments to 60.

Table 2 shows the index size of the proposed indices. We observe that the size of Search-ST and Search-CST increases linearly as the number of sequences increases. In comparison with Search-ST, the numbers of internal nodes, edges, and leaf nodes of Search-CST are reduced 50% approximately. However, because the edges of the compact subsequence tree are longer than those of the corresponding subsequence tree, Search-CST saves about 36% of storage space actually.

Fig. 6 shows the average query processing time of the three methods with various values of ε. We selected 200 sequences of average length 100 from the stock data set. The average length of query sequences was set to 20. The values of ε were determined to retrieve 10, 30, 100, and 300 answers, respectively.

As expected, the average query processing times of Search-CST and Search-ST are almost same with any value of p in the L_p-based distance function. But, Search-CST performs much better than Seq-Scan. For example, when the L_∞-based time-warping distance is used, Search-CST-L_∞ performs better than Seq-Scan-L_∞ about 50–117 times with different values of ε. Search-CST-L_2 performs better than Seq-Scan-L_2 about 26–66 times and Search-CST-L_1 performs better than Seq-Scan-L_1 about 13–23 times.

As the values of ε in L_1 and L_2 must be larger than those of L_∞ to retrieve the same number of answers,

Table 2
The size of the proposed indices with the increasing number of sequences

Number of sequences	Search algorithm	Number of edges	Number of internal nodes	Number of leaf nodes	Index size (KBytes)
200	Search-ST	997,806	997,805	819,000	44,960
	Search-CST	537,176	537,175	350,300	28,483
400	Search-ST	1,992,098	1,992,097	1,638,000	90,534
	Search-CST	1,098,474	1,098,473	723,721	58,373
600	Search-ST	3,000,515	3,000,514	2,457,000	136,101
	Search-CST	1,649,482	1,649,481	1,081,029	87,433
800	Search-ST	3,998,892	3,998,891	3,276,000	181,170
	Search-CST	2,191,398	2,191,397	1,435,043	115,903
1000	Search-ST	5,006,675	5,006,674	4,095,000	227,021
	Search-CST	2,727,899	2,727,898	1,782,489	144,326

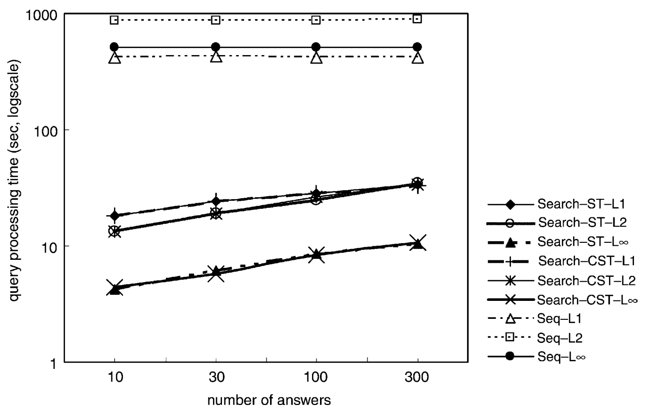

Fig. 6. Average query processing time with the increasing number of answers.

the query processing times of Search-CST-L_1 and Search-CST-L_2 get longer than that of Search-CST-L_∞.

With the results of Experiment 2, we decided that Search-CST is better than Search-ST because it solves the problem of the index size while preserving the good performance. Therefore, in the following experiments, we only compared Search-CST with the sequential scan method.

In Experiment 3, we compared the average query processing time of the two approaches while changing the average length of query sequences. We selected 200 sequences of average length 100 from the stock data set. The values of ε were determined to retrieve about 100 answers. Fig. 7 shows the average query processing time of Search-CST and Seq-Scan with the increasing length of query sequences. The result shows that Search-CST performs better than Seq-Scan regardless of query sequence length. As seen from this figure, the performance improvements of Search-CST are similar to those in Fig. 6 with any value of p in the L_p-based distance function.

In Experiment 4, we compared the two approaches with increasing number of sequences and increasing average length of sequences, respectively. We used a large volume synthetic data set for this experiment. The average length of query sequences was set to 20.

First, we fixed the length of sequences at 100 and increased the number of sequences from 2000 to 10,000. Fig. 8 shows the query processing time of the two approaches with various numbers of sequences. The elapsed times of both approaches increase linearly as

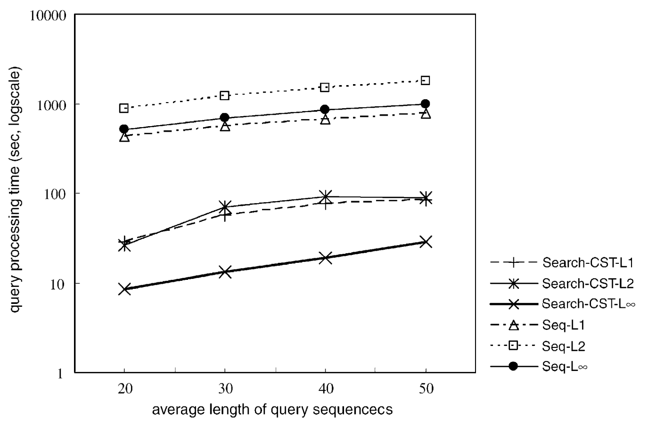

Fig. 7. Average query processing time with the increasing average length of query sequences.

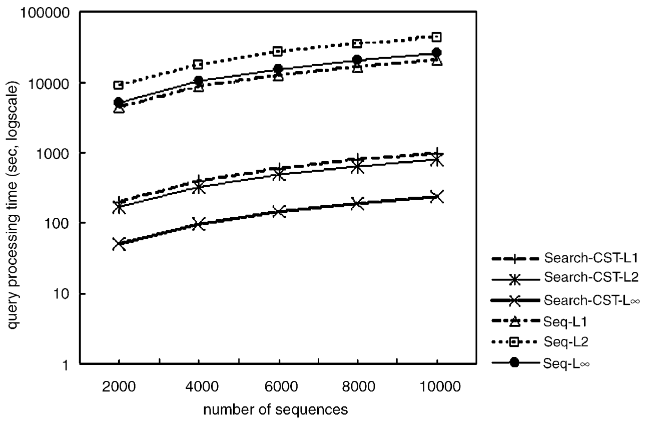

Fig. 8. Average query processing time with the increasing number of data sequences.

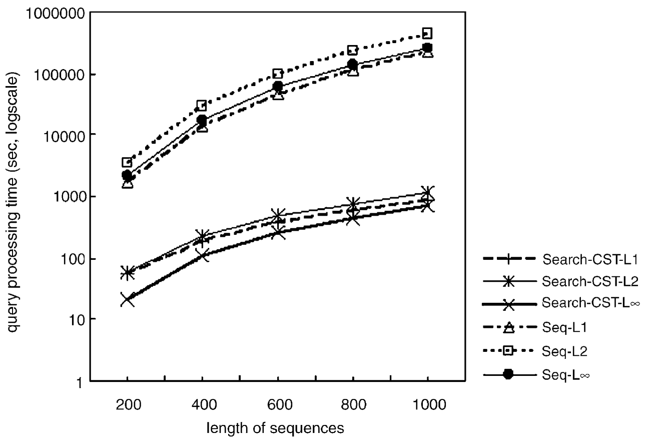

Fig. 9. Average query processing time with the increasing length of data sequences.

the number of sequences grows. The value of ε was chosen to retrieve about 100 answers from 2000 data sequences. The result shows that Search-CST shows better performance than Seq-Scan regardless of the number of sequences. As seen from this figure, the performance improvements of Search-CST are similar to those in Fig. 6 with any value of p in the L_p-based distance function.

Then, we fixed the number of sequences at 100 and increased the length of sequences from 200 to 1000. Fig. 9 shows the query processing times of the two approaches with changing average length of sequences. The value of ε was chosen to retrieve about 100 answers from data sequences of length 200. As shown in Fig. 9, while the elapsed time of Seq-Scan increases rapidly, that of Search-CST increases quite slowly. For example, when the L_∞-based time-warping distance function is used, Search-CST-L_∞ performs better than Seq-Scan-L_∞ about 102–362 times. Search-CST-L_2 performs better than Seq-Scan-L_2 about 61 to 390 times and Search-CST-L_1 performs better than Seq-Scan-L_1 about 31 to 253 times. The performance gain gets larger as the length of sequence increases.

7. Conclusions

This paper discussed the problem of shape-based retrieval in time-series databases. This paper defined a new similarity model for shape-based subsequence retrieval, and also proposed the indexing and query processing methods for supporting this similarity model efficiently.

The proposed similarity model supports a combination of transformations such as shifting, scaling, moving average, and time warping, and allows users to choose an L_p distance function to computing the similarity between the two *finally-transformed* sequences. Thus, it enables users to define target results in querying depending on their preferences.

Also, we proposed a compressed subsequence tree and a query processing method for efficient processing of the shape-based retrieval without false dismissal. The compressed subsequence tree is a compact version of a disk-based subsequence tree. An important feature of our approach is to support our similarity model based on L_1, L_2, and L_∞ with only one index structure.

To verify the superiority of our approach, we performed a series of experiments with a real-world S&P 500 stock data set and large synthetic data sets. The results reveal that our approach successfully finds all the subsequences that have the shapes similar to that of the query sequence, and also achieves several ten times to several hundred times speedup compared with the sequential scan method.

Acknowledgments

This work has been supported by Korea Research Foundation with Grant KRF-2003-041-D00486, the IT Research Center via Kangwon National University, and the University Research Program (C1-2002-146-0-3) of IITA. Sang-Wook Kim would like to thank Jung-Hee Seo, Suk-Yeon Hwang, Grace (Joo-Young) Kim, and Joo-Sung Kim for their encouragement and support.

References

Agrawal, R., Faloutsos, C., Swami, A., 1993. Efficient similarity search in sequence databases. In: Proc. FODO. pp. 69–84.

Agrawal, R., Lin, K., Sawhney, H.S., Shim, K., 1995. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In: Proc. VLDB. pp. 490–501.

Agrawal, R., Psaila, G., Wimmers, E.L., Zäit, M., 1995. Querying shapes of histories. In: Proc. VLDB. pp. 502–514.

Beckmann, N., Kriegel, H., Schneider, R., Seeger, B., 1990. The R^*-tree: an efficient and robust access method for points and rectangles. In: Proc. ACM SIGMOD. pp. 322–331.

Berndt, D.J., Clifford, J., 1996. Finding patterns in time series: a dynamic programming approach. In: Advances in Knowledge Discovery and Data Mining. AAAI/MIT, Cambridge, MA, pp. 229–248.

Chatfield, C., 1984. The Analysis of Time-series: an Introduction, third ed. Chapman and Hall, London.

Chen, M.S., Han, J., Yu, P.S., 1996. Data mining: an overview from database perspective. IEEE TKDE 8 (6), 866–883.

Chu, K.W., Wong, M.H., 1999. Fast time-series searching with scaling and shifting. In: Proc. ACM PODS. pp. 237–248.

Das, G., Gunopulos, D., Mannila, H., 1997. Finding similar time series. In: Proc. PKDD, pp. 88–100.

Faloutsos, C., Ranganathan, M., Manolopoulos, Y., 1994. Fast subsequence matching in time-series databases. In: Proc. ACM SIGMOD, pp. 419–429.

Goldin, D.Q., Kanellakis, P.C., 1995. On similarity queries for time-series data: constraint specification and implementation. In: Proc. Constraint Programming. pp. 137–153.

Kendall, M., 1979. Time-series, second ed. Charles Griffin and Company, London.

Kim, S.W., Park, S., Chu, W.W., 2001. An index-based approach for similarity search supporting time warping in large sequence databases. In: Proc. IEEE ICDE. pp. 607–614.

Loh, W.K., Kim, S.W., Whang, K.Y., 2000. Index interpolation: an approach for subsequence matching supporting normalization transform in time-series databases. In: Proc. ACM CIKM. pp. 480–487.

Loh, W.K., Kim, S.W., Whang, K.Y., 2001. Index interpolation: a subsequence matching algorithm supporting moving average transform of arbitrary order in time-series databases. IEICE Trans. Inf. Syst. E84-D (1), 76–86.

Moon, Y.S., Whang, K.Y., Loh, W.K., 2001. Duality-based subsequence matching in time-series databases. In: Proc. IEEE ICDE. pp. 263–272.

Park, S., Chu, W.W., Yoon, J., Hsu, C., 2000. Efficient searches for similar subsequences of different lengths in sequence databases. In: Proc. IEEE ICDE. pp. 23–32.

Park, S., Kim, S.W., Cho, J.S., Padmanabhan, S., 2001. Prefix-querying: an approach for effective subsequence matching under time warping in sequence databases. In: Proc. ACM CIKM. pp. 255–262.

Perng, C.S., Wang, H., Zhang, S.R., Parker, D.S., 2000. Landmarks: a new model for similarity-based pattern querying in time series databases. In: Proc. IEEE ICDE. pp. 33–42.

Preparata, F.P., Shamos, M., 1985. Computational Geometry: an Introduction. Springer-Verlag, Berlin.

Rabiner, L., Juang, H.H., 1993. Fundamentals of Speech Recognition. Prentice Hall, Englewood Cliffs, NJ.

Rafiei, D., 1999. On similarity-based queries for time series data. In: Proc. IEEE ICDE. pp. 410–417.

Rafiei, D., Mendelzon, A., 1997. Similarity-based queries for time-series data. In: Proc. ACM SIGMOD. pp. 13–24.

Shim, K., Srikant, R., Agrawal, R., 1997. High-dimensional similarity joins. In: Proc. IEEE ICDE, April. pp. 301–311.

Sidiropoulos, N.D., Bros, R., 1999. Mathematical programming algorithms for regression-based non-linear filtering in. IEEE Trans. Signal Process. (Mar).

Stephen, G.A., 1994. String Searching Algorithms. World Scientific Publishing, Singapore.

Yi, B.K., Faloutsos, C., 2000. Fast time sequence indexing for arbitrary L_p norms. In: Proc. VLDB. pp. 385–394.

Yi, B.-K., Jagadish, H.V., Faloutsos, C., 1998. Efficient retrieval of similar time sequences under time warping. In: Proc. IEEE ICDE. pp. 201–208.