Shape-Based Retrieval of Similar Subsequences in Time-Series Databases

Sang-Wook Kim
Department of Computer, Information, and Communications Engineering
Kangwon National University
wook@cc.kangwon.ac.kr

Jeehee Yoon
Department of Computer Science
Hallym University
jhyoon@sun.hallym.ac.kr

Sanghyun Park
IBM T.J. Watson Research Center
sanghyun@us.ibm.com

Tae-Hoon Kim
Department of Computer Science
Hallym University
thkim@center.cie.hallym.ac.kr

ABSTRACT

This paper deals with the problem of *shape-based retrieval* in time-series databases. The shape-based retrieval is defined as the operation that searches for the (sub)sequences whose shapes are similar to that of a given query sequence. In this paper, we propose an effective and efficient approach for shape-based retrieval of subsequences. We first introduce a new similarity model for shape-based retrieval that supports various combinations of transformations such as shifting, scaling, moving average, and time warping. For efficient processing of the shape-based retrieval, we also propose the indexing and query processing methods. To verify the superiority of our approach, we perform extensive experiments with the real-world S&P 500 stock data. 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 significant speedup over the sequential scan method.

Keywords

Similarity Search, Shape-Based Retrieval, Indexing, 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 [1]. Sequences of stock prices, money exchange rates, temperature data, product sales data, and company growth rates are the typical examples of time-series databases [2, 9]. Similarity search is an operation that finds sequences or subsequences whose changing patterns are similar to that of a given query sequence [1, 2, 9]. Similarity search is of growing importance in many new applications such as data mining and data warehousing [6, 21].

In order to measure the similarity of any two sequences of length n, most approaches [1, 7, 9, 10, 21] map the sequences into points in n-dimensional space and compute the Euclidean distance between those points as a similarity measure. However, they often fail to search for the data sequences that are actually similar to a query sequence in users' perspective when employing only the Euclidean distance. Therefore, recent work on similarity search tends to support various types of transformations such as scaling [2, 7], shifting [2, 7], normalization [8, 10, 14], moving average [21, 15], and time warping [4, 16, 17, 18, 24].

This paper discusses 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.

Agrawal et al. [3] allow a user to specify a required query pattern using their *shape definition language (SDL)*. Given a query pattern defined by SDL, they utilize 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.

Perng et.al [19] 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.

To provide a flexible solution to the above problems, this paper introduces a new similarity model that supports various combinations of transformations such as shifting, scaling, moving average, and time warping.

Similarity search is classified into *whole matching* and *subsequence matching* [1]. Subsequence matching searches for the subsequences, contained in data sequences, that are similar to a query sequence of arbitrary length. Subsequence matching is applicable to a wider range of applications than whole matching.

This paper first defines an effective similarity model for *shape-based subsequence retrieval* and presents the indexing and query processing methods for supporting this model. To verify the superiority of the approach, we perform extensive experiments with real-life stock data. 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 defines the notation and terminology used in this paper and introduces our similarity model. Section 3 presents the indexing method for supporting the proposed similarity model, and Section 4 describes our query processing method. Section 5 presents the experimental results to show the superiority of our method, and finally Section 6 summarizes and concludes the paper.

2. PROBLEM DEFINITION

This section defines the problem we are going to solve. Section 2.1 defines the notation and terminology used in this paper. Section 2.2 describes our similarity model.

2.1 Notation and Terminology

Table 1 defines the notation frequently used in this paper. A *data sequence* is a sequence stored in a database and a *query sequence* is a sequence submitted for querying.

Notation	Description
$S = (s[i])$	data sequence $(0 \leq i < Len(S))$.
$Len(S)$	number of elements in S.
$X = (x[i])$	arbitrary subsequence of S, $(0 \leq i < Len(X) \leq Len(S))$.
$Q = (q[i])$	query sequence $(0 \leq i < Len(Q))$.
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, $(s[1], s[2], ..., s[Len(S)-1])$.
$Max(S)$	the largest element of S.
$Min(S)$	the smallest element of S.
$()$	empty sequence.
$max(a, b)$	largest value between a and b.
$min(a, b, c)$	smallest value among a, b, and c.

Table 1: Notation.

Definition 1. Given a sequence $S = (s[i])$ $(0 \leq i < Len(S))$, its normalized sequence $Norm(S) = (s'[i])$ $(0 \leq i < Len(S))$ is defined as follows [2].

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

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 $S1$ and $S2$ in Figure 1. Although $S1$ and $S2$ have different element values, we see that their changing patterns are quite similar. By normalization, they are transformed into identical sequences $Norm(S1)$ and $Norm(S2)$.

Definition 2. Given a sequence $S = (s[i])$ $(0 \leq i < Len(S))$ and a moving average coefficient k, S is transformed into $MV_k(S) = (s_k[j])$ $(0 \leq j < Len(S) - k + 1)$ by k-moving average transformation [11, 5].

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

As described in Definition 2, k-moving average transformation generates a series of elements from the average values of successive k elements of an original sequence. 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, Figure 2 shows sequence S, and its 4- and 8- moving averaged sequences $MV_4(S)$ and $MV_8(S)$. We see that the effect of noises reduces as the moving average coefficient k increases.

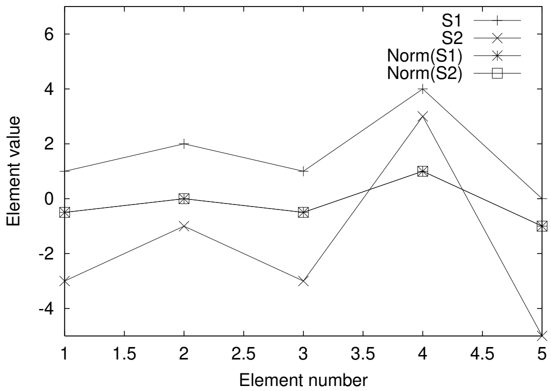

Figure 1: Example of normalization transformation.

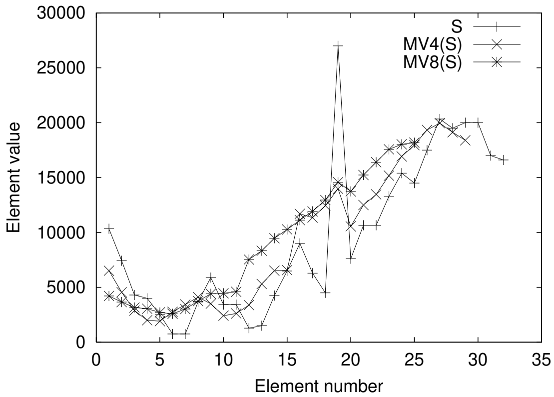

Figure 2: Example of moving average transformation.

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 [22].

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

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 [24]. 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 [13]. Here, D_{base} can be any L_p function that returns the distance of two elements.

$$\begin{aligned} D_{tw}((),()) &= 0 \\ D_{tw}(S,()) &= D_{tw}((),Q) = \infty \\ D_{tw}(S,Q) &= D_{base}(First(S), First(Q)) \\ &\quad + min \begin{cases} D_{tw}(S, Rest(Q)) \\ D_{tw}(Rest(S), Q) \\ D_{tw}(Rest(S), Rest(Q)) \end{cases} \end{aligned}$$

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.

2.2 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 supports shifting, scaling, moving average, and time warping transformations together.

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

$$D(S,Q) \;=\; D_{tw}(Norm(MV_k(S)), Norm(MV_k(Q)))$$

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. Especially, we use L_∞ as D_{base} of the time warping distance. For this similarity model, the time warping distance shown in Definition 4 is rewritten as in Definition 6.

Definition 6. Given two sequences or subsequences S and Q, their time warping distance D_{tw} is defined recursively as follows:

$$\begin{aligned} D_{tw}((),()) &= 0 \\ D_{tw}(S,()) &= D_{tw}((),Q) = \infty \\ D_{tw}(S,Q) &= max \begin{cases} |First(S) - First(Q)| \\ min \begin{cases} D_{tw}(S, Rest(Q)) \\ D_{tw}(Rest(S), Q) \\ D_{tw}(Rest(S), Rest(Q)) \end{cases} \end{cases} \end{aligned}$$

row6	**6**	4	3	**3**
row5	**6**	4	3	3
row4	**7**	4	3	4
row3	**6**	3	2	3
row2	**5**	2	1	2
row1	**4**	1	1	1
	S / Q	**3**	**4**	**3**
		column1	column2	column3

Table 2: Cumulative distance table for S = (4, 5, 6, 7, 6, 6) and Q = (3, 4, 3).

D_{tw} can be efficiently computed using a dynamic programming technique [13, 4] which builds a *cumulative distance table*[1]. For example, Table 2 shows the cumulative distance table to compute the distance of $S = (4, 5, 6, 7, 6, 6)$ and $Q = (3, 4, 3)$.

We define the problem of the shape-based retrieval in a time-series database as follows: Given a query sequence Q and a distance tolerance ε, we find the subsequences X whose distances to Q ($D(X,Q) = (D_{tw}\ (Norm(MV_k(X)), Norm(MV_k(Q)))))$ are within ε. As a result, we get the sequences S that contain X and the starting offset of X inside S.

In our similarity model, $D(X,Q) \le \varepsilon$ implies that *every element of the time-warped sequence of $Norm(MV_k(X))$ is within ε from its corresponding element of the time-warped sequence of $Norm(MV_k(Q))$*. This shape-based similarity model supports various combinations of transformations such as shifting, scaling, time warping, and moving average.

Unlike the previous methods [4, 16, 24], we employ L_∞ rather than L_1 as D_{base}. This is mainly for alleviating users' burden of specifying a tolerance [12]. Compared with L_1 or L_2, L_∞ is sensitive to noises within sequences. As stated earlier, however, our similarity model first performs moving average transformation, thereby removing the noises in each sequence. Therefore, we do not need to worry about the noise effect in applying L_∞.

3. INDEXING

This section describes the indexing method for efficient shape-based retrieval. Section 3.1 explains the suffix tree, the underlying index structure in our method. Section 3.2 discusses the indexing strategy for utilizing the suffix tree. Section 3.3 presents the index construction steps in detail.

3.1 Suffix Tree

A *trie* is a data structure for indexing a set of keywords of varying sizes. A *suffix trie* [23] is a trie whose set of keywords comprises the suffixes of a single sequence. Nodes with a

[1] Under our similarity model, this table is not a *cumulative* one any longer since we employ L_∞ instead of L_1 or L_2 as D_{base} in computing D_{tw}. However, we decide to follow the same term in the literature instead of devising a new one.

single outgoing edge can be collapsed, yielding the structure known as the *suffix tree* [23]. 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. Park et al. [16] used the suffix tree for effective subsequence matching under time warping.

Each suffix of a sequence is represented by a leaf node. For example, given sequence $S_i = (s[0], s[1], ..., s[Len(S_i)-1])$, its suffix $(s[j], s[j+1], ..., s[Len(S_i)-1])$ is represented by the leaf node labeled with (S_i, j). The edges are labeled with the subsequences such that the concatenation of the edge labels on the path from the root to the leaf labeled with (S_i, j) becomes suffix $(s[j], s[j+1], ..., s[Len(S_i)-1])$. 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. Figure 3 shows the suffix tree constructed from two sequences $S_1 = (1.1, 2.5, 2.5, 1.1)$ and $S_2 = (1.1, 2.5)$.

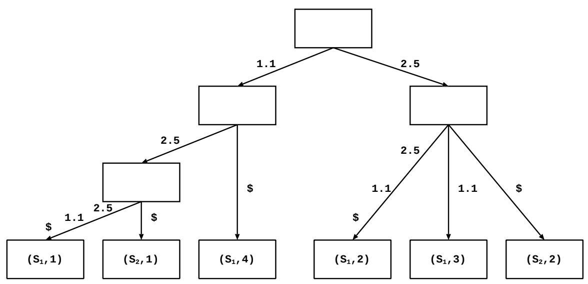

Figure 3: Suffix tree from $S_1 = (1.1, 2.5, 2.5, 1.1)$ and $S_2 = (1.1, 2.5)$.

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 of a float-type. Park et al. [16] 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.

3.2 Indexing Strategy

Consider suffix S_a of sequence S and 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 during which 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 $Max(S)$ and $Min(S)$ to normalize (sub)sequence S. Let us consider suffix S_a and its prefix S_b again. Since $Max(S_a)$ and $Min(S_a)$ are usually different from $Max(S_b)$ and $Min(S_b)$, we can not guarantee that $Norm(S_b)$ is the prefix of $Norm(S_a)$. Therefore, the distance of $Norm(S_b)$ and Q is no longer a by-product of the distance computation between $Norm(S_a)$ 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 sequences with n elements, the subsequence tree stores $\frac{n(n+1)}{2}$ subsequences. However, the subsequence tree does not grow excessively by the following reasons: 1) while the suffixes from the same sequence do not seem to have common prefixes, the subsequences from the same sequence have the high possibility to share common prefixes, and 2) normalization defined in Definition 1 makes every element take the value from the range of -1 and 1. Therefore, even though the original elements have different values, it is likely that they are converted into the identical symbol after categorization. Our preliminary experiments revealed that the number of internal nodes in the subsequence tree is almost same as the number of internal nodes in the suffix tree when we build them on the same database.

3.3 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. $MV_k(S)$ denotes the sequence converted from S by k-moving average transformation.

- **Step 2: Subsequence extraction**

  We extract every possible subsequence X from $MV_k(S)$. Note that X is not a subsequence of S but a subsequence of $MV_k(S)$. 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 subsequence tree.

- **Step 3: Normalization transformation**

  We normalize every subsequence X using $Max(X)$ and $Min(X)$. $Norm(X)$ 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 $Norm(X)$ into the symbol of the corresponding category. The number of categories depends on a target application [16].

- **Step 5: Tree construction**

  We build a subsequence tree from a set of symbol sequences. Since the subsequence tree is possibly constructed from a large volume of a database, we employ the disk-based algorithm [16] that optimizes the number of disk accesses.

4. QUERY PROCESSING

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

4.1 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 shows algorithm S-ST that traverses the subsequences tree to retrieve the subsequences whose time warping distances D_{tw} from query sequence Q are within the distance tolerance ε. 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 3.3, a 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, S-ST uses the distance function D_{tw-lb} defined in [16] to compute the lower bound distance between the query sequence and symbolized sequence CS.

Definition 7. Given query sequence Q and symbolized sequence CS, the lower-bound time warping distance function $D_{tw-lb}(CS, Q)$ is defined as follows:

$$
\begin{aligned}
D_{tw\text{-}lb}((),()) &= 0\\
D_{tw\text{-}lb}(CS,()) &= D_{tw\text{-}lb}((),Q) = \infty\\
D_{tw\text{-}lb}(CS,Q) &= max\left\{\begin{array}{l} D_{base\text{-}lb}(First(CS), First(Q))\\ min\left\{\begin{array}{l} D_{tw\text{-}lb}(CS, Rest(Q))\\ D_{tw\text{-}lb}(Rest(CS), Q)\\ D_{tw\text{-}lb}(Rest(CS), Rest(Q))\end{array}\right.\end{array}\right.
\end{aligned}
$$

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

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 S-ST calls algorithm VisitNode-and-FindAnswers to retrieve the candidate subsequences whose lower bound time warping distances from Q are within ε. When the algorithm visits node N, it inspects each child node to find new candidates and determine 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 $label(N, CN_i)$. 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() adds a new row using the distance function D_{tw-lb}. The first step is to inspect the last column of the newly added row to find candidates. If the last column has a value not larger than ε and child node CN_i is a leaf node, then the algorithm inserts the subsequence stored in CN_i into the candidate set and continues to search with the next child of N.

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, 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.

Since VisitNode-and-FindAnswers finds candidates using the lower-bound distance function D_{tw-lb}, the subsequences whose actual distances are larger than ε may be contained in the candidate set. Those subsequences are called *false alarms* [9]. Therefore, S-ST requires the post-processing step to detect and discard false alarms. For each answer in the candidate set, function PostProcess () retrieves the corresponding subsequence, transforms it using k-moving average and normalization, and then computes its 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.

Input : subsequence tree ST, query sequence Q, tolerance ε
Output: set of answers $answerSet$

candidateSet := VisitNode-and-FindAnswers (rootNode (ST), Q, ε, emptyTable);
answerSet := PostProcess (candidateSet);

return $answerSet$;

Algorithm 1: Similarity search algorithm using subsequence tree S-ST.

4.2 Algorithm Analysis

Before analyzing the complexity of S-ST, let us examine the complexity of the sequential scan method. The sequential scan method reads each data sequence S, performs k-moving average transformation for S to get $MA_k(S)$, extracts every possible subsequence X from $MA_k(S)$, and normalizes every X. Given query sequence Q and subsequence X, it builds the cumulative distance table with the computation complexity of $O(|Q||X|)$ [4]. 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 the sequential scan method is $O(ML^3|Q|)$.

S-ST is computationally much cheaper than the sequential scan method due to branch-pruning and sharing of common cumulative distance tables for all the subsequences that have common prefixes. Thus, the complexity of S-ST is $O(\frac{ML^3|Q|}{R_d R_p} + nL|Q|)$. The left term is for tree traversal and

Input : node N, query sequence Q, tolerance ε, cumulative distance table T
Output: set of candidate answers $candidateSet$

candidateSet := {};

for *each child node* CN_i *of the node* N **do**
  CT_i := AddRow (T, Q, $label(N, CN_i)$, D_{tw-lb});
  Let $dist$ be the last column value of the new row;
  if CN_i *is a leaf node and* $dist \leq \varepsilon$ **then**
    candidateSet := candidateSet ∪ FindAnswer (CN_i);
    visit the next child;
  Let $mDist$ be the minimum column value of the new row;
  if $mDist \leq \varepsilon$ **then**
    candidateSet := candidateSet ∪ VisitNode-and-FindAnswers ($CN_i, Q, \varepsilon, CTi$);

return $candidateSet$;

Algorithm 2: Index traversal algorithm VisitNode-and-FindAnswers.

the right term is for post-processing. Here, $R_d(\geq 1)$ is the reduction factor due to sharing of the cumulative distance tables, and $R_p(\geq 1)$ is the reduction factor gained from the branch-pruning. n represents the number of candidate subsequences that require the post-processing.

R_d grows as the length and the number of common prefixes increase. That is, R_d depends on the distribution of element values. Since we build the subsequence tree from the symbol sequences, we can expect a high value of R_d. R_p is dependent on the number of answers required by a user. That is, R_p increases as the distance tolerance ε decreases. If ε is so small that just one or two subsequences may be answers, only the topmost part of the subsequence tree would be touched during query processing. In the other extreme case where ε is large enough for all subsequences to be answers, all the nodes of the subsequence tree need to be visited, thus making $R_p = 1$.

5. PERFORMANCE EVALUATION

This section evaluates the effectiveness and performance of the proposed approach with S&P stock data set. The total number of stock data sequences is 545 and their average length is 231. The stock data were based on their daily closing prices. To reduce the effect of noises, we performed k-moving average transformation. As mentioned in Section 4, query sequences were randomly selected from the k-moving averaged data sequences.

Figure 4 shows the example of similar subsequences retrieved by our similarity model. We have six boxes in Figure 4. In each box, the dotted line represents an actual sequence in the S&P stock data set, and the plain line represents its 10-moving averaged sequence. The bold box located in the top and the leftmost represents the sequence selected for querying. In that sequence, we used the bold subsequence of the *double bottom pattern* as a query sequence. The remaining five boxes represent final answers obtained

# of categories	index size (Kbytes)	elapsed time (sec)
10	46,131	40.6
20	54,651	35.8
40	59,751	28.0
60	61,746	22.6
80	62,802	23.8
100	63,457	21.2

Table 3: Index sizes and average query processing times with increasing numbers of categories.

from the data set. 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.

To evaluate the performance of our approach, we examined the query processing times and index sizes in comparison with those of the sequential scan method. We selected two hundred sequences of average length 100 from the data set and applied 10-moving average transformation. The length of query sequences was set to 20. Note that in all the figures, the Y-axis is in the log scale with base-10.

Experiment 1 examined the index sizes and the average query processing times with various numbers C of categories. The distance tolerance was set to 0.11 to get 100 answers approximately. As shown in Table 3, 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 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 also set the number of categories for all the following experiments to 60.

Experiment 2 examined the query processing times with increasing values of ε. ε was determined to retrieve 10, 30, 100, and 300 answers. As shown in Figure 5, our approach was up to 35 times faster with 10 answers, and 20 times faster with 300 answers than the sequential scan method.

Experiment 3 changed the average length of query sequences and compared the query processing times of the two approaches. The values of ε were determined to retrieve about 100 answers. As shown in Figure 6, the performance gain of our approach is similar to that in Figure 5.

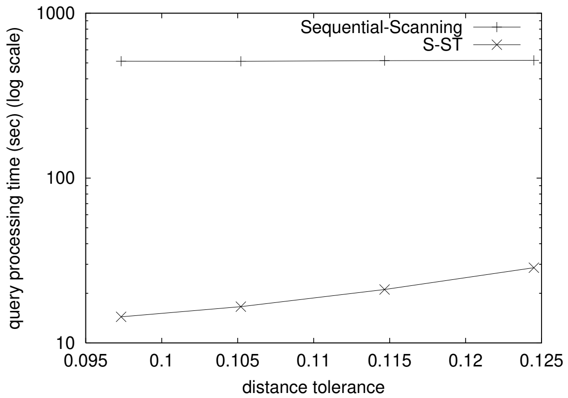

Figure 5: Comparison of query processing times with various values of the distance threshold.

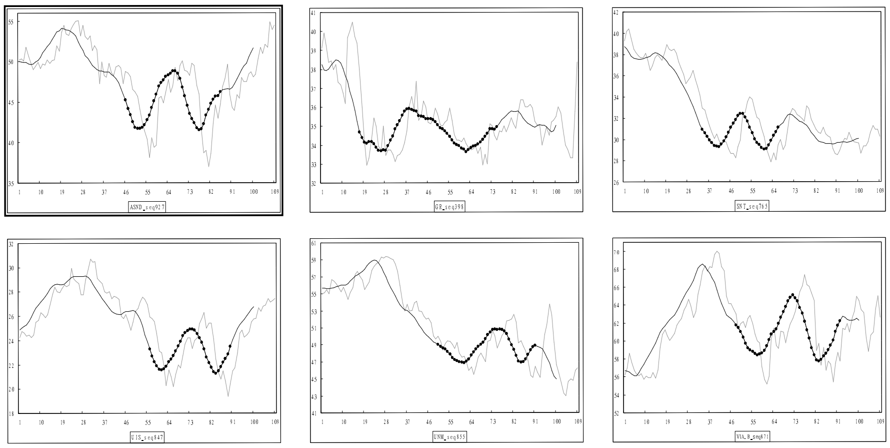

Figure 4: Example of shape-based retrieval of similar subsequences.

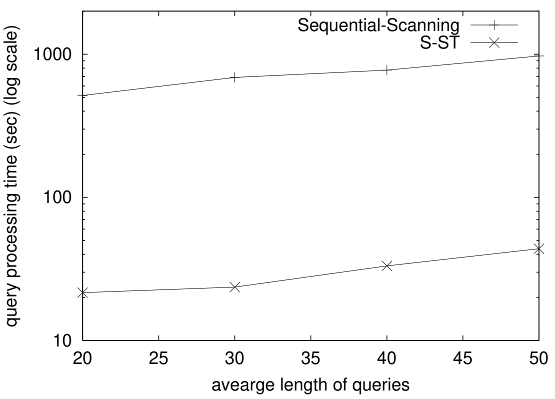

Figure 6: Comparison of query processing times with various lengths of query sequences.

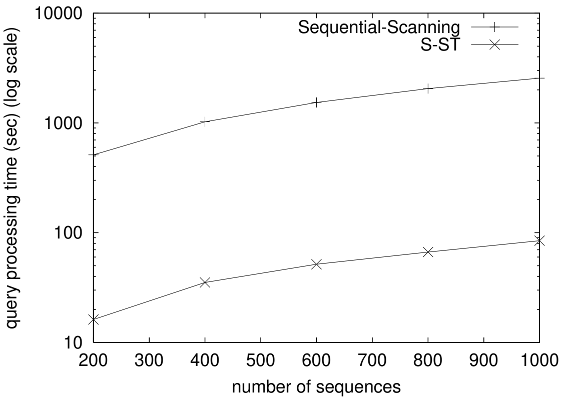

Figure 7: Comparison of query processing times with various numbers of sequences.

Experiments 4 and 5 compared the two approaches with increasing number of sequences and increasing average length of sequences, respectively. First, we fixed the length of sequences at 200 and increased the number of sequences from 200 to 1,000. Then, we fixed the number of sequences at 200 and increased the length of sequences from 100 to 200.

Experiment 4 examined the query processing times of the two approaches with various numbers of sequences. ε was chosen as 0.10, which makes about 100 answers retrieved among 800 data sequences. As shown in Figure 7, the elapsed times of both approaches increase linearly as the number of sequences grows, but the slope of our approach is much less than that of the sequential scan method. S-ST achieves more than 30 times speedup in all cases.

Experiment 5 examined the query processing times of the two approaches with changing average length of sequences. ε was chosen as 0.14, which has about 100 answers retrieved among data sequences of length 160. As shown in Figure 8, while the elapsed time of the sequential scan method increases rapidly, that of S-ST increases quite slowly. Compared with the sequential scan method, S-ST has the performance gain of 25 times to 66 times, and the performance gain gets larger as the length of sequences increases.

6. 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 presented the indexing and query processing methods for supporting this similarity model.

The proposed similarity model supports various combinations of transformations such as shifting, scaling, moving average, and time warping, and thus enables effective shape-based retrieval. There have been several works that support each transformation separately, but there has been no approach that combines all these transformations.

For efficient shape-based retrieval without the occurrence of any false dismissal, this paper also proposed a method for disk-based construction of a subsequence tree and a method for query processing based on the tree.

To verify the superiority of our approach, we performed a series of experiments with the real-world S&P 500 stock data set. 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 up to 66 times speedup up compared with the sequential scan method.

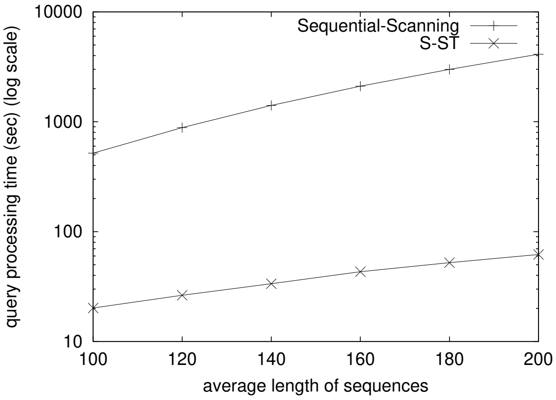

Figure 8: Comparison of query processing times with various average lengths of sequences.

Acknowledgments: This research was supported by the 2000 Basic Research Program (Grant 2000-1-51200-006-1) of KOSEF and also partially by the BK21 Program at Kangwon National University. Sang-Wook Kim would like to thank Jung-Hee Seo, Suk-Yeon Hwang, Grace(Ju-Young) Kim, and Ju-Sung Kim for their encouragement and support.

7. REFERENCES

[1] R. Agrawal, C. Faloutsos, A. Swami, "Efficient Similarity Search in Sequence Databases", *Proc. FODO*, pp. 69-84, 1993.

[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", *Proc. VLDB*, pp. 490-501, 1995.

[3] R. Agrawal, G. Psaila, E. L. Wimmers, M. Zäit, "Querying Shapes of Histories", *Proc. VLDB*, pp. 502-514, 1995.

[4] D. J. Berndt, J. Clifford, "Finding Patterns in Time Series: A Dynamic Programming Approach", *Advances in Knowledge Discovery and Data Mining*, AAAI/MIT, pp. 229-248, 1996.

[5] C. Chatfield, *The Analysis of Time-Series: An Introduction*, 3rd Edition, Chapman and Hall, 1984.

[6] M. S. Chen, J. Han, P. S. Yu, "Data Mining: An Overview from Database Perspective", *IEEE TKDE*, Vol. 8, No. 6, pp. 866-883, 1996.

[7] K. W. Chu, M. H. Wong, "Fast Time-Series Searching with Scaling and Shifting", *Proc. ACM PODS*, pp. 237-248, 1999.

[8] G. Das, D. Gunopulos, H. Mannila, "Finding Similar Time Series", *Proc. PKDD*, pp. 88-100, 1997.

[9] C. Faloutsos, M. Ranganathan, Y. Manolopoulos, "Fast Subsequence Matching in Time-Series Databases", *Proc. ACM SIGMOD*, pp. 419-429, 1994.

[10] D. Q. Goldin, P. C. Kanellakis, "On Similarity Queries for Time-Series Data: Constraint Specification and Implementation", *Proc. Constraint Programming*, pp. 137-153, 1995.

[11] M. Kendall, *Time-Series*, 2nd Edition, Charles Griffin and Company, 1979.

[12] S. W. Kim, S. Park, W. W. Chu, "An Index-Based Approach for Similarity Search Supporting Time Warping in Large Sequence Databases", *Proc. IEEE ICDE*, pp. 607-614, 2001.

[13] L. Rabiner, H. H. Juang, *Fundamentals of Speech Recognition*, Prentice Hall, 1993.

[14] W. K. Loh, S. W. Kim, K. Y. Whang, "Index Interpolation: An Approach for Subsequence Matching Supporting Normalization Transform in Time-Series Databases", *Proc. ACM CIKM*, pp. 480-487, 2000.

[15] W. K. Loh, S. W. Kim, K. Y. Whang, "Index Interpolation: A Subsequence Matching Algorithm Supporting Moving Average Transform of Arbitrary Order in Time-Series Databases", *IEICE Trans. on Information and Systems*, Vol. E84-D, No. 1, pp. 76-86, Jan. 2001.

[16] S. Park, W. W. Chu, J. Yoon, C. Hsu, "Efficient Searches for Similar Subsequences of Different Lengths in Sequence Databases", *Proc. IEEE ICDE*, pp. 23-32, 2000.

[17] S. Park, S. W. Kim, W. W. Chu, "Segment-Based Approach for Subsequence Searches in Sequence Databases", *Proc. ACM SAC*, pp. 248-252, 2001.

[18] S. Park, S. W. Kim, J. S. Cho, S. Padmanabhan, "Prefix-Querying: An Approach for Effective Subsequence Matching Under Time Warping in Sequence Databases", *Proc. ACM CIKM*, 2001. (accepted to appear)

[19] C. S. Perng, H. Wang, S. R. Zhang, D. S. Parker, "Landmarks: a New Model for Similarity-based Pattern Querying in Time Series Databases", *Proc. IEEE ICDE*, pp. 33-42, 2000.

[20] F. P. Preparata, M. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, 1985.

[21] D. Rafiei, A. Mendelzon, "Similarity-Based Queries for Time-Series Data", *Proc. ACM SIGMOD*, pp. 13-24, 1997.

[22] K. Shim, R. Srikant, R. Agrawal, "High-dimensional Similarity Joins", *Proc. ICDE*, pp. 301-311, Apr. 1997.

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

[24] B.-K. Yi, H. V. Jagadish, C. Faloutsos, "Efficient Retrieval of Similar Time Sequences Under Time Warping", *Proc. IEEE ICDE*, pp. 201-208, 1998.