Efficient Processing of Similarity Search Under Time Warping in Sequence Databases: An Index-Based Approach [⋆]

Sang-Wook Kim [a,*] Sanghyun Park [b] Wesley W. Chu [c]

[a] *College of Information and Communications*
Hanyang University, Korea

[b] *Department of Computer Science and Engineering*
Pohang University of Science and Technology (POSTECH), Korea

[c] *Department of Computer Science*
University of California at Los Angeles (UCLA), USA

Abstract

This paper discusses the effective processing of similarity search that supports time warping in large sequence databases. Time warping enables sequences with similar patterns to be found even when they are of different lengths. Prior methods for processing similarity search that supports time warping failed to employ multi-dimensional indexes without false dismissal since the time warping distance does not satisfy the triangular inequality. They have to scan the entire database, thus suffering from serious performance degradation in large databases. Another method that hires the suffix tree, which does not assume any distance function, also shows poor performance due to the large tree size.

In this paper, we propose a novel method for similarity search that supports time warping. Our primary goal is to enhance the search performance in large databases without permitting any false dismissal. To attain this goal, we have devised a new distance function, D_{tw-lb}, which consistently underestimates the time warping distance and satisfies the triangular inequality. D_{tw-lb} uses a 4-tuple feature vector that is extracted from each sequence and is invariant to time warping. For the efficient processing of similarity search, we employ a multi-dimensional index that uses the 4-tuple feature vector as indexing attributes, and D_{tw-lb} as a distance function. We prove that our method does not incur false dismissal. To verify the superiority of our method, we have performed extensive experiments. The results reveal that our method achieves a significant improvement in speed up to 43 times faster with a data set containing real-world S&P 500 stock data sequences, and up to 720 times with data sets containing a very large volume of synthetic data sequences. The performance gain increases: (1) as the number of data sequences increases, (2) the average length of data sequences increases, and (3) as the tolerance in a query decreases. Considering the characteristics of real databases, these tendencies imply that our approach is suitable for practical applications.

Key words: Similarity search, Sequence database, Indexing, Time warping distance

1 Introduction

The sequence 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 sequence databases [2,3]. Similarity search is an operation that finds sequences or subsequences whose changing patterns are similar to that of a given query sequence [1–3]. Similarity search is of growing importance in many new applications such as data mining and data warehousing [4,5].

Similarity search is classified into whole matching and subsequence matching [1]. Assuming that all the data and query sequences have the same length, whole matching searches for the data sequences similar to a query sequence. Subsequence matching searches for the subsequences, contained in data sequences, that are similar to a query sequence of arbitrary length.

In order to measure the similarity of any two sequences of length n, most approaches [1,6,3,7,5] map the sequences into points in an 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 the users' perspective when employing only the Euclidean distance as a similarity measure. Therefore, recent work on similarity search tends to support various types of transformations such as scaling [2,6,7], shifting [2,6,7], normalization [8,7], moving average [9,5], and time warping [10–12].

Time warping is a transformation that allows any sequence element to replicate itself as many times as needed without extra costs [12]. For example, two sequences $S = \langle 20, 21, 21, 20, 20, 23, 23, 23 \rangle$ and $Q = \langle 20, 20, 21, 20, 23 \rangle$ can be identically transformed into $\langle 20, 20, 21, 21, 20, 20, 23, 23, 23 \rangle$ by time warping. The *time warping distance* is defined as the smallest distance between two sequences transformed by time warping. While the Euclidean distance can only be used when the two sequences compared are of the same length, the time warping distance can be applied to any two sequences of arbitrary lengths.

⋆ Recommended by Dr. Nick Koudas (koudas@research.att.com)

∗ Contact author
Email address: `wook@ihanyang.ac.kr` (Sang-Wook Kim).

Therefore, the time warping distance fits well with databases where sequences are of different lengths [1]. The time warping distance is widely used in such applications as voice recognition [13] and electro-cardiogram analysis.

For the efficient processing of similarity search, most current approaches [1–3] employ multi-dimensional indexes [14–16]. Yi et al. [12] claimed that the multi-dimensional indexes assuming the *triangular inequality* [17] directly or indirectly cause *false dismissal* in similarity search when their distance functions do not satisfy the triangular inequality. False dismissal [1,3] is to miss part of the final query result. Yi et al. [12] proved that the time warping distance does not satisfy the triangular inequality. Therefore, the multi-dimensional indexes that assume the triangular inequality could not work with the time warping distance in many applications that do not permit false dismissal.

For guaranteeing no false dismissal, Berndt et at. [10] and Yi et. al [12] proposed similarity searching techniques that support time warping without using indexes. Since these techniques must scan all the data sequences sequentially to perform similarity search, their performance degrades seriously in large databases. For resolving this performance degradation, Yi et al. [12] also proposed a method that maps a sequence of arbitrary length n into a k-dimensional point ($k < n$) using the FastMap [18], a feature extraction function, and then builds a multi-dimensional index on a set of mapped points. By using a multi-dimensional index, this method improves the search performance significantly. However, this method is only suited for restricted applications because it cannot avoid false dismissal. Park et al. [11] proposed an efficient method for similar subsequence matching under time warping by using the *suffix tree* [19] as its index structure. This method guarantees no false dismissal since the suffix tree does not assume any distance function. However, this method failed in providing a systematic guideline to perform optimal *categorization*, which is the prerequisite to achieve high search performance. Furthermore, it suffers from serious performance degradation in whole matching due to the large size of its suffix tree.

This paper proposes a new effective method for similarity search that supports time warping. Our primary goal is to enhance the search performance in large databases without permitting any false dismissal. To attain this goal, we have devised a new distance function, D_{tw-lb}, which consistently underestimates the time warping distance and also satisfies the triangular inequality. As in-

[1] Sequences of different lengths need to be compared in the following situations: (1) when sequences have different sampling rates, for example, one sequence may be sampled every minute while another sequence is sampled every second; and (2) when sequences have different starting points, for example, a sequence may start today while another sequence began a year ago. The time warping distance is very useful in these situations. Instead of focusing individual elements of sequences, the time warping distance compares their fluctuation patterns along the time axis.

put parameters, D_{tw-lb} uses a 4-tuple feature vector that is extracted from each sequence and is invariant to time warping. For the efficient processing of similarity search, we employ a four-dimensional index that uses the 4-tuple feature vector as indexing attributes, and D_{tw-lb} as a distance function. To the extent of the authors' knowledge, our method is the first index-based approach to similarity search under time warping that uses a distance function. We prove that our method does not incur false dismissal. To verify the superiority of our method, we have performed extensive experiments. The results reveal that our method outperforms the previous ones by up to 43 times when using a data set containing real-world S&P 500 stock data sequences, and up to 720 times when using data sets containing a very large volume of synthetic data sequences.

This paper is organized as follows. Section 2 defines the notation and terminology used in this paper. Section 3 briefly reviews previous methods for similarity search supporting time warping and points out their problems. Section 4 proposes the strategy and the algorithms of our method in detail. Section 5 evaluates the performance of the proposed method through extensive experiments in comparison with the previous ones. Finally, Section 6 summarizes and concludes the paper.

2 Notation and Terminology

A sequence $S(= \langle s_1, s_2, ..., s_{|S|}\rangle)$ is an ordered list of elements. $|S|$ is the length of S and s_i is its i^{th} element. $First(S)$ and $Last(S)$ are the first and last elements of S, respectively. $Rest(S)$ is a subsequence of S that includes the elements from position 2 to the end. That is, $Rest(S) = \langle s_2, ..., s_{|S|}\rangle$. $\langle\rangle$ denotes an empty sequence. Sequences stored in a database are called data sequences and a sequence submitted as a query is called a query sequence. This paper focuses on sequences of numeric elements.

Different similarity measures have been discussed in the literature. Among others, L_p function is the most popular class of similarity measures. With the requirement that S and Q have the same length $(= L)$, L_p function is defined as follows:

$$L_p(S, Q) = \sqrt[p]{\textstyle\sum_{i=1}^{L} |s_i - q_i|^p}, \quad 1 \leq p \leq \infty.$$

It is called the *city-block* or the *Manhattan* distance when $p = 1$, and the *Euclidean* distance when $p = 2$. In the case when $p = \infty$, it is called the *maximum* distance [20], and can be reformulated as follows:

$$L_\infty(S, Q) = \max_{i=1}^{L} |s_i - q_i|$$

Now, let us describe the time warping distance. Time warping is a generalization of classical algorithms for comparing discrete sequences with sequences of continuous values [13]. To find the minimum difference between two sequences, time warping enables each element of a sequence to match one or more neighboring elements of the other sequence.

Definition 1: Given two sequences S and Q, the time warping distance D_{tw} is defined recursively as follows:

$$
\begin{aligned}
D_{tw}(\langle\rangle, \langle\rangle) &= 0 \\
D_{tw}(S, \langle\rangle) &= D_{tw}(\langle\rangle, Q) = \infty \\
D_{tw}(S, Q) &= \sqrt[p]{(L_p(First(S), First(Q)))^p \; + \; (MinStutterDistance(S,Q))^p} \\
&= \sqrt[p]{|First(S) - First(Q)|^p \; + \; (MinStutterDistance(S,Q))^p} \\
MinStutterDistance(S,Q) &= min \begin{cases} D_{tw}(S, Rest(Q)) \\ D_{tw}(Rest(S), Q) \\ D_{tw}(Rest(S), Rest(Q)) \end{cases}
\end{aligned}
$$

■

In databases where sequences are of different lengths, the data sequences whose time warping distances to a query sequence are smaller than a given tolerance are considered similar. Thus, *similarity search supporting time warping* is defined as the problem of searching for such data sequences from a database.

3 Related Work

This section briefly summarizes the previous methods for similarity search that supports time warping, and points out their problems.

3.1 Naive-Scan

There have been many research efforts on similarity search supporting time warping for voice recognition [13]. The method in this area reads all the data sequences and computes D_{tw} between a data sequence S and a query sequence Q using the dynamic programming technique [10,13] based on the *recurrence relation* $\gamma_{tw}(x, y)$.

Definition 2: Given two sequences S and Q, the recurrence relation $\gamma_{tw}(x, y)$ $(x = 1, 2, ..., |S|,\ y = 1, 2, ..., |Q|)$ that builds the *time warping distance table* for S

and Q is defined as follows [10]:

$$\gamma_{tw}(x,y) \;=\; \sqrt[p]{|First(S) - First(Q)|^p \;+\; (min\ (\gamma_{tw}(x,y-1),\ \gamma_{tw}(x-1,y),\ \gamma_{tw}(x-1,y-1)))^p}$$

■

The dynamic programming algorithm fills in the time warping distance table as the computation proceeds. The final distance, $\gamma_{tw}(|S|,|Q|)$, is the minimum distance between S and Q, and the matching of elements can be traced backward in the table by choosing the previous cells with the lowest distance. This method incurs a high CPU cost since the dynamic programming technique has the time complexity of $O(|S|*|Q|)$ to compute $D_{tw}(S,Q)$. Furthermore, this method must scan the entire database, thus its performance degenerates seriously in large databases. We refer to this method as *Naive-Scan*.

3.2 LB-Scan

Yi et al. [12] tried to solve the problem of similarity search supporting time warping in the database perspective, and proposed two methods to improve the performance of Naive-Scan. This method uses a distance function D_{lb} to discard non-qualifying sequences without false dismissal [1]. Let $max(S)$ and $max(Q)$ denote the maximum values in S and Q, respectively. Similarly, $min(S)$ and $min(Q)$ denote the minimum values in S and Q. With L_1 as a base distance function, D_{lb} is defined in Definition 3.

Definition 3: Given two sequences S and Q, the distance function $D_{lb}(S,Q)$ that returns the lower-bound distance of $D_{tw}(S,Q)$ is defined as follows [12]:

$$D_{lb}(S,Q) \;=\; min \begin{cases} \sum_{s_i > max(Q)} |s_i - max(Q)| \;+\; \sum_{q_j < min(S)} |q_j - min(S)| & \text{if } S \text{ and } Q \text{ overlap} \\ \sum_{s_i > max(Q)} |s_i - max(Q)| \;+\; \sum_{s_i < min(Q)} |s_i - min(Q)| & \text{if } S \text{ encloses } Q \\ max\ (\sum_{i=1}^{|S|} |s_i - max(Q)|,\ \sum_{j=1}^{|Q|} |q_j - min(S)|) & \text{if } S \text{ and } Q \text{ are disjoint} \end{cases}$$

■

Definition 3 assumes $max(S) > max(Q)$. D_{lb} saves CPU cost because the definition of D_{lb} requires just one scan of each sequence. That is, the $D_{lb}(S,Q)$ is computed in the $O(|S| + |Q|)$ time complexity. However, this method still has to scan all data sequences from the database to perform similarity search; therefore, it is not suitable for large database environments. We call this method *LB-Scan*.

3.3 FastMap Method

The other method proposed in [12] is the *FastMap* method. This method first maps a sequence of length n into a k-dimensional point using the FastMap [18]. Given a set of N sequences and a distance function, the FastMap maps the sequences into N points in a k-dimensional space, so that the original distances among those sequences are preserved well. The key idea of the FastMap is to pretend as if sequences of length n are indeed points in an n-dimensional space, and to try to project these points on k mutually orthogonal directions, using only the distance information.

Next, this method builds a multi-dimensional index on a set of mapped k-dimensional points. Using the L_1 distance function as a filter on the multi-dimensional index, the method efficiently searches for the candidate sequences that are possibly included in the final query result. This method highly reduces both CPU and disk access costs. Yi et al. [12] also proposed a method that combines the FastMap method and LB-Scan in a pipelined manner.

As addressed in the paper, the critical problem of the FastMap method is that it does not guarantee no false dismissal. Thus, this method is not applicable to many applications that do not permit false dismissal. In addition, it is not trivial to choose the number k of dimensions that provides the best search performance in actual situations.

3.4 ST-Filter

To resolve the shortcomings of the FastMap method, Park et al. [11] proposed a method called ST-Filter that uses the *suffix tree* [19] as an index structure. The suffix tree was originally proposed to find the substrings that are exactly matched to a given query string.

To make the index structure compact and thus accelerate query processing, ST-Filter converts sequences of numeric elements to sequences of symbols via *categorization*, and builds a suffix tree from the converted sequences. Park et al. [11] also proposed to use the sparse suffix tree that stores a subset of suffixes whose first values are different from their preceding values. ST-Filter does not cause any false dismissal since (1) it does not assume any distance function during index construction, and (2) it employs lower-bound distance functions to filter out dissimilar subsequences during query processing.

ST-Filter was originally designed for subsequence matching. The performance gain of the suffix tree comes from common subsequences contained in the database. As the number and the average length of common subsequences increase, ST-Filter shows better performance. The merit of the suffix tree diminishes when applied to whole matching, however. In this case, the performance gain comes only from a small number of common *prefixes*, making ST-Filter much less effective.

The performance of ST-Filter depends on the number of categories, however, it is difficult to determine the optimal number of categories. As the number of categories increases, the number of candidate subsequences decreases while the suffix tree gets larger due to the reduced number of common subsequences. Thus, ST-Filter has a big trade-off between the candidate access and suffix tree access costs depending on the granularity of categories. Currently, we are investigating a systematic guideline to determine the optimal number of categories by analyzing the characteristics of the target database.

4 The Proposed Method

This section proposes a new method for similarity search supporting time warping. Section 4.1 introduces and justifies our similarity model. Section 4.2 presents our indexing strategy, and Section 4.3 describes the algorithms for index construction and query processing.

4.1 Similarity Model

The time warping distance is defined as in Definition 1 in the references [21,13,12], where the *base distance function* is not restricted to a specific L_p function. Users can choose any L_p function in the definition depending on the characteristics of their applications [12].

For any two sequences, we measure their similarity using the time warping distance that employs L_∞ as a base distance function. For this similarity model, the time warping distance shown in Definition 1 is rewritten as in Definition 4.

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

$$\begin{aligned}
D_{tw}(\langle\rangle,\langle\rangle) &= 0\\
D_{tw}(S,\langle\rangle) &= D_{tw}(\langle\rangle,Q) = \infty\\
D_{tw}(S,Q) &= \sqrt[\infty]{(L_\infty(First(S),First(Q)))^\infty \ + \ (MinStutterDistance(S,Q))^\infty}\\
&= \sqrt[\infty]{|First(S)-First(Q)|^\infty \ + \ (MinStutterDistance(S,Q))^\infty}\\
&= max\begin{cases} |First(S)-First(Q)| \\ min\begin{cases} D_{tw}(S,Rest(Q)) \\ D_{tw}(Rest(S),Q) \\ D_{tw}(Rest(S),Rest(Q)) \end{cases} \end{cases}
\end{aligned}$$

■

Let $M = \langle m_1, m_2, ..., m_{|M|} \rangle$ be the best element mappings that obtain the minimum difference between S and Q ($|S| \leq |M|$, $|Q| \leq |M|$). Then, each element mapping is represented as $m_h = (s_i, q_j)$ ($1 \leq h \leq |M|$, $1 \leq i \leq |S|$, $1 \leq j \leq |Q|$). The distance of each element mapping is computed as $|m_h| = |s_i - q_j|$. Since our similarity model employs L_∞ as a base distance function, the distance of S and Q is simply expressed as $D_{tw}(S, Q) = max\ (|m_1|, |m_2|, ..., |m_{|M|}|)$. Given a query sequence Q and a tolerance ϵ, similarity search finds all data sequences S that satisfy the following condition: $D_{tw}(S, Q) \leq \epsilon$. If $D_{tw}(S, Q) \leq \epsilon$, it is obvious that every element of S is within ϵ from its corresponding element of Q.

Unlike the previous methods [10–12], we employ L_∞ rather than L_1 as a base distance function for alleviating users' burden of specifying a tolerance. L_1 causes the time warping distance to be highly affected by the lengths of the time-warped sequences. This makes it difficult to decide an appropriate tolerance at querying time. By using L_∞, however, users can easily specify an appropriate tolerance without considering the lengths.

L_∞ provides an extra gain on search performance. Since the time warping distance requires a large CPU cost, we need to discard non-qualifying sequences as early as possible. In the case of L_1, such decisions happen when the distance accumulated exceeds a tolerance. In the case of L_∞, on the other hand, the decisions happen each time the distance between any element pair exceeds a tolerance. As a result, the time warping distance with L_∞ incurs a CPU cost much less than that with L_1.

The time warping distance equipped with L_1 has been widely used in applications such as voice recognition and electro-cardiogram analysis. In applications other than voice recognition and electro-cardiogram, however, there has been no dominant L_p function used as a base distance function. The applications of our similarity model may include stock price analysis, money exchange-rate analysis, company growth-rate analysis, product sales analysis, and temperature analysis. We are currently using the L_∞ function for stock price analysis, and also observed that it is quite useful due to the reasons mentioned above.

Our similarity model is also useful in detecting the identity or high similarity of any two sequences of different lengths where one of them has been transformed from another due to noise. As an example, let us consider an advanced application that tries to detect illegal copies of video clips on the web. A video clip consists of a series of frames. To speed up the content-based similarity search in video databases, we usually extract a set of representative features from each frame [22,23]. As a result, we easily map a video clip into a single-dimensional numeric sequence when we extract a single feature (e.g., an average gray-level) from each frame. Some noise may be accidentally or intentionally introduced into video clips due to format or resolution changes. Frame rates are also possibly changed (i.e., 30 frames per second vs. 15 frames per second) to reduce the network traffic. To detect similar video clips in such an environment, we first determine the noise threshold. Then, we apply our similarity model to compute similarities between a video clip given in a query and those video clips on the web in order to retrieve candidates. Lastly, we obtain the final result by manually performing a thorough inspection of the candidates. Similarly, detecting illegal copies of music files can also benefit from our similarity model.

4.2 Indexing Strategy

The goal of our method is to guarantee **no false dismissal** as well as **high search performance**. Most indexes implicitly or explicitly assume that their distance function satisfies the triangular inequality. When employing a distance function that does not satisfy the triangular inequality, it causes false dismissal in searching. Yi et al. [12] proved that the time warping distance does not satisfy the triangular inequality and claimed that the only method guaranteeing no false dismissal is the sequential scan. However, it is well known that the sequential scan suffers from severe performance degradation in large database environments.

For resolving the problem, we have devised a new distance function D_{tw-lb} that consistently lower-bounds D_{tw} and also satisfies the triangular inequality. For the efficient processing of similarity search without false dismissal, we build a multi-dimensional index that employs D_{tw-lb} as a distance function. As input parameters, D_{tw-lb} requires the features of each sequence. The feature extraction under time warping is not easy since time warping possibly transforms a sequence into new multiple sequences of various lengths and patterns depending on a query sequence. The features of a data sequence, however, have to be unique independently of a given query sequence, as well as time warping, since they would play a role in a multi-dimensional index as indexing attributes. Thus, the feature extraction must be represented as a form of a function.

From each sequence S stored in a database, we extract a 4-tuple feature vector, $Feature(S) = \langle First(S), Last(S), Greatest(S), Smallest(S)\rangle$. $First(S)$ and $Last(S)$ are the first and the last elements of S, and $Greatest(S)$ and $Smallest(S)$ are the greatest and the smallest elements of S. Since time warping stretches sequences along the time axis, this feature vector is invariant to time warping with any query sequence. The feature extraction performs with the complexity of $O(|S|)$. Using the feature vector, we define a lower-bound function D_{tw-lb} that consistently returns a distance smaller than or equal to D_{tw} as follows:

Definition 5: Given two sequences S and Q, the lower-bound distance function $D_{tw-lb}(S, Q)$ is defined as:

$$D_{tw-lb}(S,Q) \;=\; L_\infty(Feature(S), Feature(Q)) \;=\; max \begin{cases} |First(S) - First(Q)| \\ |Last(S) - Last(Q)| \\ |Greatest(S) - Greatest(Q)| \\ |Smallest(S) - Smallest(Q)| \end{cases}$$

■

Theorem 1: For any two sequences $S = \langle s_1, ..., s_n \rangle$ and $Q = \langle q_1, ..., q_m \rangle$, the following inequality always holds.

$$D_{tw}(S,Q) \;\;\geq\;\; D_{tw-lb}(S,Q)$$

■

Proof: Let S' and Q' be such sequences that are time-warped from S and Q in order to minimize the time warping distance of S and Q. Then, $|S'| = |Q'| = k$ ($|S| \leq k, |Q| \leq k$). Let us first consider the following derivation.

$$\begin{aligned} D_{tw}(S,Q) &= L_\infty(S', Q') \\ &= L_\infty(\langle s'_1, ..., s'_k \rangle,\ \langle q'_1, ..., q'_k \rangle) \\ &= max\ (L_\infty(\langle s'_2, ..., s'_{k-1} \rangle,\ \langle q'_2, ..., q'_{k-1} \rangle),\ L_\infty(\langle s'_1, s'_k \rangle,\ \langle q'_1, q'_k \rangle)) \\ &= max\ (L_\infty(\langle s'_2, ..., s'_{k-1} \rangle,\ \langle q'_2, ..., q'_{k-1} \rangle),\ L_\infty(\langle First(S), Last(S) \rangle,\ \langle First(Q), Last(Q) \rangle)) \\ &\geq L_\infty(\langle First(S), Last(S) \rangle,\ \langle First(Q), Last(Q) \rangle) \end{aligned}$$

By the above derivation, we have proved that Equation 4.1 is always true for any two sequences S and Q.

$$D_{tw}(S,Q) \;\;\geq\;\; L_\infty(\langle First(S), Last(S) \rangle,\ \langle First(Q), Last(Q) \rangle) \;\; (4.1)$$

The next step to prove Theorem 1 is to verify Equation 4.2 for any two sequences S and Q.

$$D_{tw}(S,Q) \;\;\geq\;\; L_\infty(\langle Greatest(S), Smallest(S) \rangle,\ \langle Greatest(Q), Smallest(Q) \rangle) \;\; (4.2)$$

Let the element of Q' matched to $Greatest(S')$ be $Greatest\_Match(Q')$, and let the element of Q' matched to $Smallest(S')$ be $Smallest\_Match(Q')$. Likewise, let the element of S' matched to $Greatest(Q')$ be $Greatest\_Match(S')$, and let the element of S' matched to $Smallest(Q')$ be $Smallest\_Match(S')$.

S' and Q' have the three possible arrangements of ranges, as shown in Figure 1. For an easier explanation, let us assume that $Greatest(S') \geq Greatest(Q')$. If this assumption is not valid, the roles of S' and Q' can be exchanged. Let us consider each arrangement of ranges.

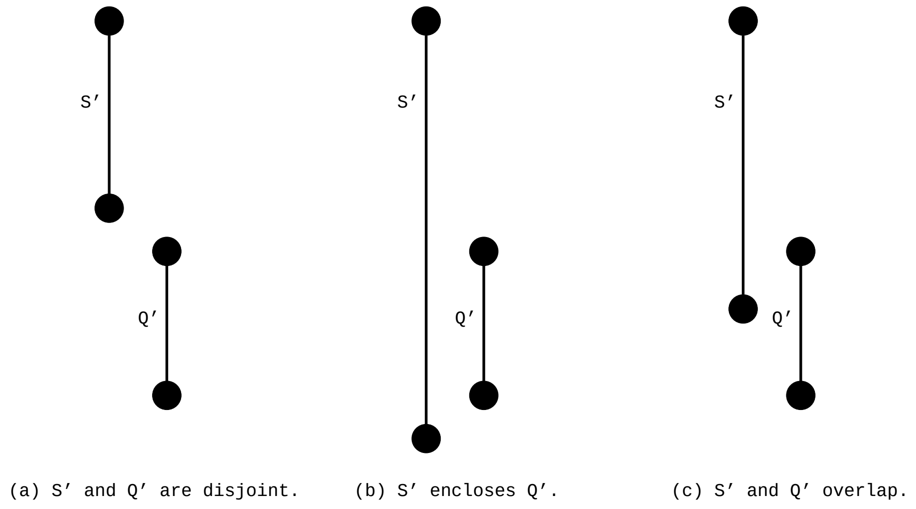

Fig. 1. Three possible arrangements of ranges of S' and Q'.

Case 1: when S' and Q' are disjoint

$$\begin{aligned}
D_{tw}(S,Q) &= L_\infty(S',Q') \\
&\geq max\ (|Greatest(S') - Greatest\_Match(Q')|,\ |Smallest(Q') - Smallest\_Match(S')|) \\
&\geq max\ (|Greatest(S') - Greatest(Q')|,\ |Smallest(Q') - Smallest(S')|) \\
&= L_\infty(\langle Greatest(S'), Smallest(S')\rangle,\ \langle Greatest(Q'), Smallest(Q')\rangle) \\
&= L_\infty(\langle Greatest(S), Smallest(S)\rangle,\ \langle Greatest(Q), Smallest(Q)\rangle)
\end{aligned}$$

Case 2: when S' encloses Q'

$$\begin{aligned}
D_{tw}(S,Q) &= L_\infty(S',Q') \\
&\geq max\ (|Greatest(S') - Greatest\_Match(Q')|,\ |Smallest(S') - Smallest\_Match(Q')|) \\
&\geq max\ (|Greatest(S') - Greatest(Q')|,\ |Smallest(S') - Smallest(Q')|) \\
&= L_\infty(\langle Greatest(S'), Smallest(S')\rangle,\ \langle Greatest(Q'), Smallest(Q')\rangle) \\
&= L_\infty(\langle Greatest(S), Smallest(S)\rangle,\ \langle Greatest(Q), Smallest(Q)\rangle)
\end{aligned}$$

Case 3: when S' and Q' overlap

$$\begin{aligned}
D_{tw}(S,Q) &= L_\infty(S',Q') \\
&\geq max\ (|Greatest(S') - Greatest\_Match(Q')|,\ |Smallest(Q') - Smallest\_Match(S')|) \\
&\geq max\ (|Greatest(S') - Greatest(Q')|,\ |Smallest(Q') - Smallest(S')|) \\
&= L_\infty(\langle Greatest(S'), Smallest(S')\rangle,\ \langle Greatest(Q'), Smallest(Q')\rangle) \\
&= L_\infty(\langle Greatest(S), Smallest(S)\rangle,\ \langle Greatest(Q), Smallest(Q)\rangle)
\end{aligned}$$

Since Equation 4.2 is satisfied for every possible arrangement of ranges of S' and Q', Equation 4.2 has been verified. Using Equation 4.1 and 4.2, the following derivation process proves Theorem 1.

$$\begin{aligned}
D_{tw}(S,Q) &\geq max \begin{cases} L_\infty(\langle First(S), Last(S)\rangle,\ \langle First(Q), Last(Q)\rangle) \\ L_\infty(\langle Greatest(S), Smallest(S)\rangle,\ \langle Greatest(Q), Smallest(Q)\rangle) \end{cases} \\
&= L_\infty \begin{pmatrix} \langle First(S), Last(S), Greatest(S), Smallest(S)\rangle, \\ \langle First(Q), Last(Q), Greatest(Q), Smallest(Q)\rangle \end{pmatrix} \\
&= L_\infty(Feature(S), Feature(Q)) \\
&= D_{tw-lb}(S,Q)
\end{aligned}$$

■

We can easily derive Corollary 1 from Theorem 1. Corollary 1 implies that similarity search that uses D_{tw-lb} rather than D_{tw} in order to discard dissimilar sequences does not incur false dismissal.

Corollary 1: For any two sequences $S = \langle s_1, ..., s_n\rangle$ and $Q = \langle q_1, ..., q_m\rangle$, and a tolerance ϵ, the following statement always holds.

$$D_{tw}(S,Q) \leq \epsilon \implies D_{tw-lb}(S,Q) \leq \epsilon$$

■

Theorem 2: For any three sequences X, Y, and Z, the following inequality always holds.

$$D_{tw-lb}(X,Z) \quad \leq \quad D_{tw-lb}(X,Y) \ + \ D_{tw-lb}(Y,Z).$$

■

Proof: Since $D_{tw-lb}(X,Z) = L_\infty(Feature(X), Feature(Z))$ and the distance function L_∞ is a metric that satisfies the triangular inequality [17], Theorem 2 is always true. ■

Theorem 2 implies that the similarity search supporting time warping with a multi-dimensional index does not suffer from false dismissal any longer by employing D_{tw-lb} as the distance function. In summary, using Corollary 1 and Theorem 2, we can safely design a novel method for similarity search supporting time warping, which guarantees no false dismissal as well as high performance.

4.3 Algorithms

This subsection presents the algorithms for index construction and query processing in our similarity searching method.

4.3.1 Index Construction

Each data sequence is mapped into a point in a 4-dimensional space since a 4-tuple feature vector $Feature(S)$ is extracted from a sequence S for indexing. For indexing a set of 4-dimensional points, any multi-dimensional index such as the R-tree [24], R$^+$-tree [16], R*-tree [14], and X-tree [15] can be used. The index construction algorithm first makes an entry $\langle$ $First(S)$, $Last(S)$, $Greatest(S)$, $Smallest(S)$, $ID(S)$ $\rangle$ for each data sequence S, and then inserts it into a multi-dimensional index. Here, $ID(S)$ is the identifier of S. If there are a large number of data sequences at the stage of initial index construction, we can achieve high performance in construction by using bulk loading methods [25–27].

4.3.2 Query Processing

Algorithm 1 shows TW-Sim-Search, our query processing algorithm. Step 1 extracts a 4-tuple feature vector from the query sequence. Step 2 performs a square-range query on a four-dimensional index. The range query uses a 4-tuple feature vector obtained in Step 1 as a central point and ϵ as a range for each dimension. The query also uses D_{tw-lb} for filtering. That is, Step 2 finds all the data points contained in $\langle [First(Q)-\epsilon, First(Q)+\epsilon], [Last(Q)-\epsilon, Last(Q)+\epsilon], [Greatest(Q)-\epsilon, Greatest(Q)+\epsilon], [Smallest(Q)-\epsilon, Smallest(Q)+\epsilon] \rangle$. Step 3 constructs a candidate set from the entries returned by Step 2. Step 4, Step 5, and Step 6 perform post-processing to discard *false alarm* [1,3]. For each entry in the candidate set, Step 5 reads a corresponding data sequence S from the database and Step 6 computes the distance between S and Q using D_{tw}. If the distance is within ϵ, it inserts S into the answer set.

Even when applications require L_1 or L_2 as a base distance function in D_{tw}, we are also able to apply our index-based approach in the same way. In such cases, we just need to use 1 or 2 instead of ∞ for p in Definition 1 for D_{tw-lb}.

Algorithm 1: Query Processing Algorithm TW-Sim-Search

```
Input  : query sequence $Q$, tolerance $\epsilon$
Output: set of data sequences $S$ satisfying $D_{tw}(S, Q) \leq \epsilon$

$answerSet := \{\}$;
1 Get a feature vector from the query sequence $Q$, $Feature(Q) = \langle\ First(Q), Last(Q), Greatest(Q), Smallest(Q)\ \rangle$ ;
2 Perform a range query on the multi-dimensional index using $Feature(Q)$ and $\epsilon$ ;
3 Make a candidate set from the entries returned by Step 2;
4 for each entry in the candidate set do
5     Read a corresponding sequence $S$ from the database;
6     if $D_{tw}(S, Q) \leq \epsilon$ then
          insert $S$ into $answerSet$
7 return $answerSet$;
```

5 Performance Evaluation

This section presents the experimental results for performance evaluation of the proposed method. Section 5.1 describes the experiment environment and the evaluation methodology, and Section 5.2 introduces and analyzes experimental results.

5.1 Evaluation Environment and Methodology

Hardware and Software Platform: The hardware platform for the experiments is the SunSparc Ultra-5 workstation equipped with 640MB RAM and 9GB hard disk with 9.5 ms seek time. The software platform is SunOS 5.8, which is set to single-user mode to minimize the interference from other system and user processes.

Data Sets: Two kinds of data sets were used for experiments: a synthetic data set and a data set containing real stock market data. Each synthetic data sequence $S = \langle s_1, .., s_n\rangle$ was generated by the following random walk expression:

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

Here, z_i is an independent, identically distributed (IID) random variable that takes values in the range $[-0.1, 0.1]$. The value of the first element s_1 is taken randomly in the range $[1, 10]$. Stock data sequences were extracted from USA S&P 500 (http://biz.swcp.com/stocks). The total number of stock data sequences is 545 and their average length is 231.

Methods Compared: We compared TW-Sim-Search, our method, with the previous ones: Naive-Scan [10], LB-Scan [12], and ST-Filter [11]. For fair evaluation, we modified all the previous methods slightly such that they employed L_∞ instead of L_1 as a base distance function for the time warping distance. For the best performance of ST-Filter, we generated 100 categories using the equal-length-interval method [11]. We excluded the FastMap method [12] from our performance evaluation since it has the deficiency of causing false dismissal. For TW-Sim-Search, we employed the R-tree [24] as a multi-dimensional index, and set its page size to 1KB.

Evaluation Methodology: We performed three experiments to evaluate the proposed method from the different perspectives. For each experiment, we executed 100 queries with query sequences generated as follows: (1) select a random sequence from the database; (2) take a random value from an appropriate range [2] for every element; and (3) add the value to the element.

The first experiment measures the *candidate ratios* of the four methods. The purpose of this experiment is to compare the filtering effects of the different methods by examining the trend of false alarm [1,3] in each method. The candidate ratio is defined as follows:

$$candidate\ ratio = \frac{the\ number\ of\ candidate\ sequences}{the\ number\ of\ data\ sequences}$$

The good filtering effect does not always result in good performance since the time spent in the filtering stage may be different depending on the filtering methods employed. Therefore, the second experiment compares the different methods, in terms of their elapsed times, with the same sets of data and query sequences as in the first experiment to evaluate their overall performance.

The final experiment compares the performance of the four methods using a large volume of synthetic data sets. The purpose of this experiment is to show the scalability of the proposed method. This scalability test is further classified into the test with a large number of data sequences, and the test with long data sequences.

5.2 Results and Analyses

In Experiment 1, the candidate ratios of all the methods were compared using the S&P 500 stock data sequences whose number is 545. The number of candidate sequences is different according to the method used to perform the similarity search. Candidate sequences are those returned after traversing the suffix tree (in ST-Filter), performing a range query on the R-tree (in TW-Sim-Search), and applying D_{lb} to all data sequences (in LB-Scan). Since the filtering stage is not required in Naive-Scan, data sequences contained in the final query result are depicted in the graph as candidate sequences.

Figure 2 depicts the results showing the candidate ratios of the four methods. The X-axis represents the tolerance and the Y-axis represents the candidate ratio. Since

[2] The range is $[\frac{-std}{10}, \frac{std}{10}]$, where *std* is the standard deviation of the sequence.

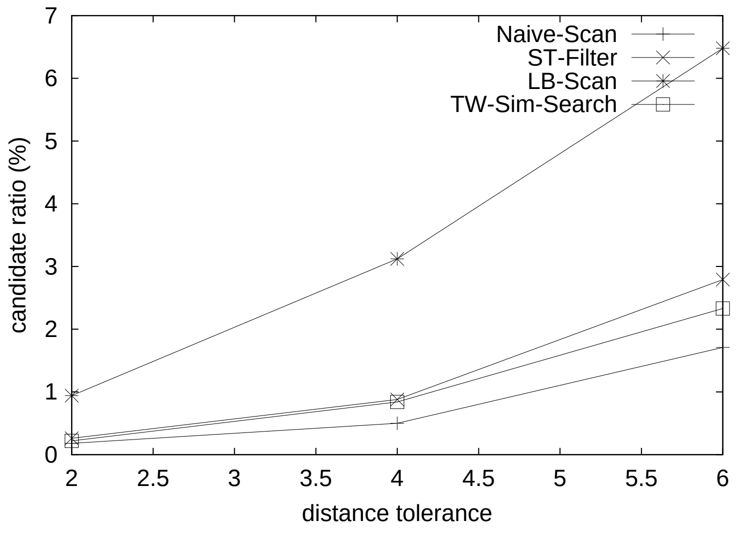

Fig. 2. Comparison of filtering effects using S&P 500 stock data sequences.

Naive-Scan does not produce false alarm, the number of candidate sequences is identical to the number of sequences in the final query result. Notice that a method whose candidate ratio is closer to that of Naive-Scan has a better filtering effect. We see that the number of sequences in the final query result varies from 1.1 ($\approxeq 0.2\%$) to 9.3 ($\approxeq 1.7\%$) depending on a tolerance. The results reveal that TW-Sim-Search has a filtering effect slightly better than ST-Filter, which is much better than LB-Scan.

The good filtering effect does not always result in good search performance since the time spent in the filtering stage is different depending on the methods employed. In the filtering stage, Naive-Scan and LB-Scan have to read all the data sequences stored in the database while ST-Filter and TW-Sim-Search have to traverse the suffix tree and the R-tree, respectively. Therefore, Experiment 2 compared the four methods in terms of their elapsed times with the same sets of data and query sequences as in Experiment 1 to evaluate their overall performance.

Figure 3 shows the elapsed times of the four methods. The X-axis represents the tolerance and the Y-axis represents the elapsed time. ST-Filter shows a worse performance even than Naive-Scan. As pointed out in Section 3.4, this is mainly because ST-Filter was designed only for subsequence matching. When employed for whole matching, however, it loses its performance gains due to the enlarged suffix tree caused by the reduced number of common subsequences.

As expected, LB-Scan shows better performance than Naive-Scan. Though both LB-Scan and Naive-Scan access all the data sequences in the database, LB-Scan could save CPU cost by using the lower-bound function D_{lb} rather than D_{tw}. Since the S&P 500 data set is not that large (about 850 KB), the reduced CPU cost appears clearly in Figure 3.

TW-Sim-Search accesses just a small portion of the R-tree, whose size is less than 4% of that of the database size, and obtains a small number of candidate sequences. As a result, TW-Sim-Search achieved improvements in speed by 4 times to 43 times,

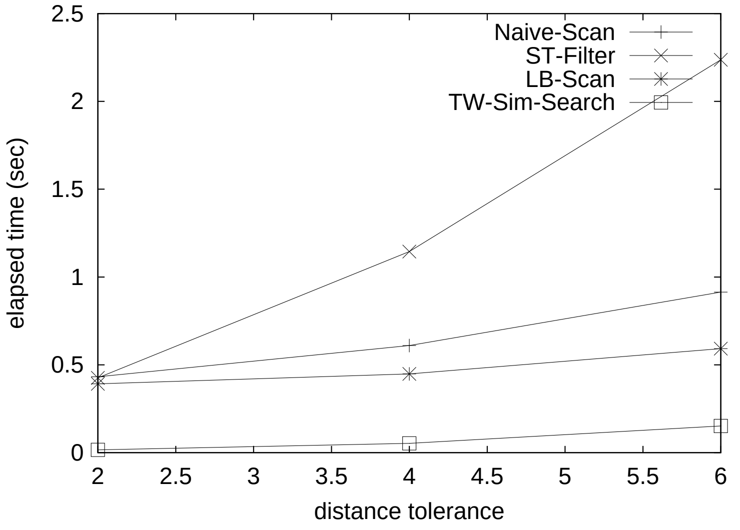

Fig. 3. Comparison of elapsed times using S&P 500 stock data sequences.

depending on the tolerance compared with LB-Scan, the best one among the previous methods. We note that the performance gain of TW-Sim-Search increases as tolerance decreases. This property is quite desirable because most users are interested in a small number of answers.

The S&P 500 data set with only 545 data sequences is not large enough to test the scalability of the four methods. Therefore, in the following experiments, we compared the performance of the four methods using a large volume of synthetic data sets.

Experiment 3 increased the number of data sequences from 1,000 to 100,000 while fixing the tolerance and the average length of data sequences at 0.1 and 1,000, respectively. Figure 4 shows the elapsed time of the four methods. Both the X-axis and the Y-axis are in the log scale with base-10.

The results show that the elapsed time of Naive-Scan, LB-Scan, and ST-Filter increases dramatically as the number of data sequences increases. The reason for this is that they scan the entire database (in cases of Naive-Scan and LB-Scan) or traverse the large suffix tree. It is very interesting that ST-Filter performs worse than Naive-Scan and LB-Scan with 1,000 and 10,000 data sequences, yet, performs better with 100,000 sequences. This is because the number of common subsequences increases as a database gets larger.

Unlike the other three methods, the performance of TW-Sim-Search appears to be nearly constant regardless of the number of data sequences. For selected numbers of data sequences, TW-Sim-Search outperforms the best one among the previous methods 19 times to 720 times with different number of data sequences. We also observe that such performance gains increase as the number of data sequences increases. This observation implies that TW-Sim-Search accesses a small portion of the R-tree, irrespective of the number of data sequences.

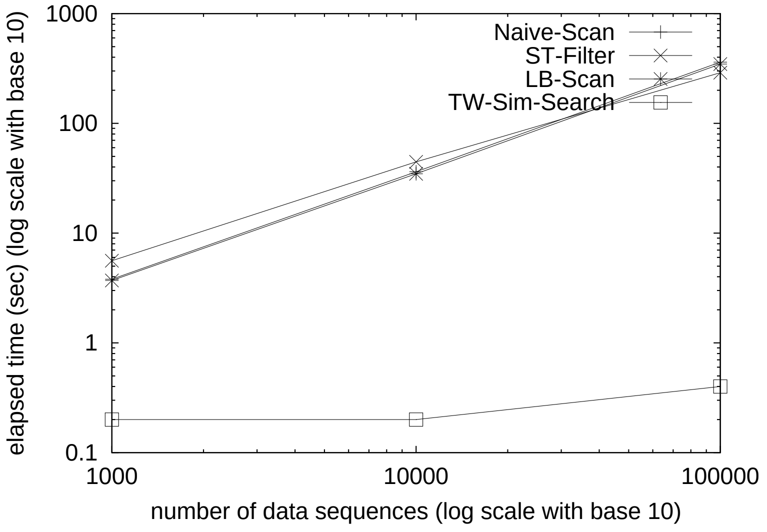

Fig. 4. Comparison of elapsed times using synthetic data with different numbers of sequences.

Finally, Experiment 4 increased the length of data sequences from 100 to 5,000, while fixing the tolerance and the number of data sequences at 0.1 and 10,000, respectively. Figure 5 shows the elapsed time of the four methods.

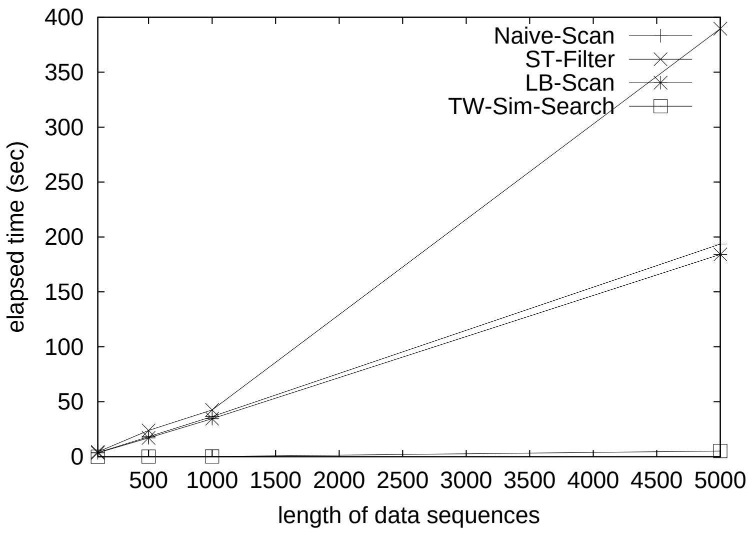

Fig. 5. Comparison of elapsed times using synthetic data sequences with different lengths.

As in Experiment 3, while the elapsed time of Naive-Scan, LB-Scan and ST-Filter grows rapidly as the length of data sequences increases, the elapsed time of TW-Sim-Search remains relatively unchanged. Compared with LB-scan, which is faster than Naive-Scan and ST-Filter in this experiment, TW-Sim-Search gains about 36 to 175 times speedup. The performance gain increases with the growing length of data sequences.

6 Concluding Remarks

Similarity search is an operation that finds data sequences whose changing patterns are similar to that of a query sequence [1,3], and is of growing importance in such applications as data mining and data warehousing [4,5]. Time warping is a useful transformation in such situations where the Euclidean distance is not applicable, since the sequences to be compared are of different lengths [11,12]. In this paper, we have discussed an efficient approach for similarity search that supports the time warping transformation.

The time warping distance does not satisfy the triangular inequality [12]. Thus, previous methods for similarity search that supports time warping could not work with multi-dimensional indexes without false dismissal since those indexes implicitly and explicitly assume the triangular inequality [12]. Berndt et al. [10] and Yi et al. [12] proposed methods that reduce the CPU cost. However, they have to scan the entire database, and are not suitable for large database environments. Park et al. [11] suggested a method that uses the suffix tree, which does not assume any distance function, to reduce the number of data sequences to be accessed. While this method fits well with subsequence matching, it shows poor performance in whole matching due to the abnormally enlarged suffix tree. Yi et al. [12] also proposed another method using the FastMap that highly accelerates the search performance. However, this method suffers from a critical problem that it does not guarantee no false dismissal.

In this paper, we have proposed a novel method that supports high search performance without false dismissal. The proposed method extracts feature vectors that are invariant to time warping from each sequence and uses D_{tw-lb}, a new lower-bound distance function of the time warping distance, to compute the distance between any two feature vectors. The proposed method efficiently processes similarity search by using a multi-dimensional index built on the set of feature vectors. For showing the robustness of our approach, we have proved that the proposed method does not incur any false dismissal. To the best of the authors' knowledge, our method is the first method supporting time warping without false dismissal that employs the index structures assuming a distance function.

For performance evaluation, we have compared our method with previous ones through extensive experiments. The results show that our method outperforms the best method among previous ones around 4 to 43 times when using the S&P stock data set, and 19 to 720 times when using a large volume of synthetic data sets. The performance gain becomes larger: (1) as the number of data sequences increases, (2) the average length of data sequences gets longer, and (3) as the tolerance in a query decreases. Considering the characteristics of real databases, these tendencies imply that our approach is suitable for practical applications.

Our method is easily applicable to subsequence matching. It builds the same index on the feature vectors from subsequences rather than whole sequences. It also applies the same algorithm for query processing. In subsequence matching, we expect much more gains in search performance, since our method performs better with a larger number of (sub)sequences to be stored in the database.

7 Acknowledgement

This research was supported by the 2000-2001 Research Project(Grant KRF-2000-041-E00258) of Korea Research Foundation(KRF) and also by the Brain Korea 21 Project in 2003. 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

[1] R. Agrawal, C. Faloutsos, A. Swami, Efficient similarity search in sequence databases, in: Proc. Int'l Conf. on Foundations of Data Organization and Algorithms (FODO), 1993, pp. 69–84.

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

[3] C. Faloutsos, M. Ranganathan, Y. Manolopoulos, Fast subsequence matching in time-series databases, in: Proc. ACM Int'l Conf. on Management of Data (SIGMOD), 1994, pp. 419–429.

[4] M. S. Chen, J. Han, P. S. Yu, Data mining: An overview from database perspective, IEEE Transactions on Knowledge and Data Engineering (TKDE) 8 (6) (1996) 866–883.

[5] D. Rafiei, A. Mendelzon, Similarity-based queries for time series data, in: Proc. ACM Int'l Conf. on Management of Data (SIGMOD), 1997, pp. 13–24.

[6] K. W. Chu, M. H. Wong, Fast time-series searching with scaling and shifting, in: Proc. ACM Symposium on Principles of Database Systems (PODS), 1999, pp. 237–248.

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

[8] G. Das, D. Gunopulos, H. Mannila, Finding similar time series, in: Proc. Principles and Practice of Knowledge Discovery in Databases (PKDD), 1997, pp. 88–100.

[9] W. K. Loh, S. W. Kim, K. Y. Whang, Index interpolation: An approach for subsequence matching supporting normalization transform in time-series

databases, in: Proc. ACM Conf. on Information and Knowledge Management (CIKM), 2000, pp. 480–487.

[10] D. J. Berndt, J. Clifford, Finding patterns in time series: A dynamic programming approach, in: Advances in Knowledge Discovery and Data Mining, AAAI/MIT, 1996, pp. 229–248.

[11] S. Park, W. W. Chu, J. Yoon, C. Hsu, Efficient searches for similar subsequences of different lengths in sequence databases, in: Proc. IEEE Int'l Conf. on Data Engineering (ICDE), 2000, pp. 23–32.

[12] B.-K. Yi, H. V. Jagadish, C. Faloutsos, Efficient retrieval of similar time sequences under time warping, in: Proc. IEEE Int'l Conf. on Data Engineering (ICDE), 1998, pp. 201–208.

[13] L. Rabinar, B.-H. Juang, Fundamentals of Speech Recognition, Prentice Hall, 1993.

[14] N. Beckmann, H. Kriegel, R. Schneider, B. Seeger, The R*-tree: An efficient and robust access method for points and rectangles, in: Proc. ACM Int'l Conf. on Management of Data (SIGMOD), 1990, pp. 322–331.

[15] S. Berchtold, D. A. Keim, H. Kriegel, The X-tree: An index structure for high-dimensional data, in: Proc. Int'l Conf. on Very Large Data Bases (VLDB), 1996, pp. 28–39.

[16] T. K. Sellis, N. Roussopoulos, C. Faloutsos, The R^+-tree: A dynamic index for multi-dimensional objects, in: Int'l Conf. on Very Large Data Bases (VLDB), 1987, pp. 507–518.

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

[18] C. Faloutsos, K. Lin, Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets, in: Proc. ACM Int'l Conf. on Management of Data (SIGMOD), 1995, pp. 163–174.

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

[20] K. Shim, R. Srikant, R. Agrawal, High-dimensional similarity joins, in: Proc. IEEE Int'l Conf. on Data Engineering (ICDE), 1997, pp. 301–311.

[21] J. R. Deller, J. G. Proakis, J. H. L. Hansen, Discrete-Time Processing of Speech Signals, Macmillan Publishing Company, 1993.

[22] S. Park, J. S. Cho, K. H. Hyun, Indexing technique for similarity matching in large video databases, in: Proc. SPIE Conf. on Storage and Retrieval for Media Databases, 2002, pp. 214–222.

[23] R. Tusch, H. Kosch, L. Boszormeny, VIDEX: An integrated generic video indexing approach, in: Proc. ACM Multimedia, 2000, pp. 448–451.

[24] A. Guttman, R-trees: A dynamic index structure for spatial searching, in: Proc. ACM Int'l Conf. on Management of Data (SIGMOD), 1984, pp. 47–57.

[25] J. Bercken, B. Seeger, P. Widmayer, A general approach to bulk loading multidimensional index structures, in: Proc. Int'l Conf. on Very Large Data Bases (VLDB), 1997, pp. 406–415.

[26] I. Kamel, C. Faloutsos, On packing R-trees, in: Proc. ACM Conf. on Information and Knowledge Management (CIKM), 1993, pp. 490–499.

[27] S. T. Leutenegger, J. M. Edgington, M. A. Lopez, STR: A simple and efficient algorithm for R-tree packing, in: Proc. IEEE Int'l Conf. on Data Engineering (ICDE), 1997, pp. 497–506.