ICDE 2001 Paper-ID : 207

An Index-Based Approach for Similarity Search Supporting Time Warping in Large Sequence Databases

Sang-Wook Kim
Software Tools and Techniques Team
IBM T.J. Watson Research Center
swkim@us.ibm.com

Sanghyun Park
Department of Computer Science
University of California, Los Angeles
shpark@cs.ucla.edu

Wesley W. Chu
Department of Computer Science
University of California, Los Angeles
wwc@cs.ucla.edu

Abstract

This paper discusses an effective processing of similarity search that supports time warping in large sequence databases. Time warping enables finding sequences with similar patterns even when they are of different lengths. Previous methods for processing similarity search that supports time warping fail to employ multi-dimensional indexes without false dismissal since the time warping distance does not satisfy the triangular inequality. They have to scan all the database, thus suffer 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 new novel method for similarity search that supports time warping. Our primary goal is to innovate on search performance in large databases without permitting any false dismissal. To attain this goal, we devise a new distance function D_{tw-lb} that consistently underestimates the time warping distance and also 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 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 perform extensive experiments. The results reveal that our method achieves significant speedup up to 43 times with real-world S&P 500 stock data and up to 720 times with very large synthetic data. The performance gain becomes larger: (1) as the number of data sequences gets larger, (2) the average length of data sequences gets longer, and (3) as the tolerance in a query gets smaller. Considering the characteristics of real databases, these tendencies imply that our approach is suitable for practical applications.

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, 11]. Similarity search is an operation that finds sequences or subsequences whose changing patterns are similar to that of a given query sequence [1, 2, 11]. Similarity search is of growing importance in many new applications such as data mining and data warehousing [7, 21].

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 of approaches [1, 8, 11, 12, 21] 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 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, 8], shifting [2, 8], normalization [9, 12, 16], moving average [17, 21], and time warping [4, 18, 25].

Time warping is a transformation that allows any sequence element to replicate itself as many times as needed without extra costs [25]. 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 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 warping distance fits well with the databases where sequences are of different lengths.[1] The time warping distance is widely used in such applications as voice recognition [20] and electro-cardiogram analysis.

For efficient processing of similarity search, most of previous approaches [1, 2, 11] employ multi-dimensional indexes [3, 5, 22]. Yi et al. [25] claimed that the multi-dimensional indexes assuming the triangular inequality [19] directly or indirectly cause *false dismissal* in similarity search when their distance functions do not satisfy the triangular inequality. False dismissal [1, 11] is to miss part of the final query result. Yi et al. [25] proved that the time warping distance does not satisfy

[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.

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. [4] and Yi et. al [25] proposed similarity searching techniques that support time warping without using indexes. Since these techniques have to 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. [25] also proposed a method that maps a sequence of arbitrary length n into a k-dimensional point ($k < n$) using the FastMap [10], 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 search performance significantly. However, this method is only applicable to restricted applications since it cannot avoid false dismissal. Park et al. [18] proposed an efficient method for subsequence search under time warping by using the suffix tree [24] as its index structure. This method guarantees no false dismissal since the suffix tree does not assume any distance function. However, this method fails to provide 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 its large size of the suffix tree.

This paper proposes a new effective method for similarity search that supports time warping. Our primary goal is to innovate on search performance in large databases without permitting any false dismissal. To attain this goal, we devise a new distance function D_{tw-lb} that consistently underestimates the time warping distance and also satisfies the triangular inequality. As input parameters, D_{tw-lb} uses a 4-tuple feature vector that is extracted from each sequence and is invariant to time warping. For 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 perform extensive experiments. The results reveal that our method outperforms the previous ones up to 43 times when using real-world stock data and up to 720 times when using very large synthetic data.

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 own problems. Section 4 proposes the strategy and the algorithms of our method in detail. Section 5 evaluates the performance of the proposed method through experiments in comparisons 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 order 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 the 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.

L_p function has been widely used to measure the similarity of two sequences S and Q. L_1 is the Manhattan distance, L_2 is the Euclidean distance, and L_∞ is the maximum distance in any pair of elements [23]. L_p function requires that two sequences to be compared have the same length.

$$L_p(S,Q) = \left(\sum_{i=1}^{|S|} |s_i - q_i|^p\right)^{1/p}, \quad 1 \le p \le \infty.$$

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 [20]. 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 [20]:

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

■

Here, D_{base} can be any L_p function that returns the distance of two elements. 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 [20]. 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 [4, 20]. This method incurs high CPU cost since the dynamic programming technique has the time complexity of $O(|S| * |Q|)$. Furthermore, this method has to scan the entire database, thus its performance degenerates seriously in large databases. We call this method *Naive-Scan* in this paper.

3.2 LB-Scan

Yi et al. [25] 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 hires a distance function D_{lb} to discard non-qualifying sequences without false dismissal [1]. D_{lb} saves CPU cost due to its reduced time complexity of $O(|S| + |Q|)$. 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 environment. We call this method *LB-Scan*.

3.3 FastMap Method

The other one proposed in [25] is the *FastMap* method. This method maps a sequence of length n into a k-dimensional point using the FastMap [10], a feature-extraction function, and builds a multi-dimensional index on a set of mapped k-dimensional points. Using the multi-dimensional index, this method efficiently searches for the candidate sequences that are possibly included in the final query result. This method highly reduces both of CPU and disk access costs. Yi et al. [25] 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 not able to 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 of dimensions k that provides the best search performance in actual situations.

3.4 ST-Filter

To resolve the shortcomings of the FastMap method, Park et al. [18] proposed a method called ST-Filter that uses the suffix tree [24] as an index structure. The suffix tree provides fast accesses to the strings that are exactly matched to a query string. ST-Filter converts sequences of numeric elements to sequences of symbols via *categorization* and builds a suffix tree from the converted sequences. ST-Filter does not cause any false dismissal since the suffix tree does not assume any distance function.

ST-Filter was designed focusing on 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. 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 not easy 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 3.1 introduces and justifies our similarity model. Section 3.2 presents our indexing strategy, and Section 3.3 describes the algorithms for index construction and query processing.

4.1 Similarity Model

For any two sequences, we measure their similarity using the time warping distance that hires L_∞ as D_{base}. For this similarity model, the time warping distance shown in Definition 1 is rewritten as in Definition 2.

Definition 2: Given two sequences S and Q, their time warping distance D_{tw} is defined recursively as follows:

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

■

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 by D_∞ in our similarity model. That is, $|m_h| = |s_i - q_j|$. As a result, 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 the 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 [4, 18, 25], we employ L_∞ rather than L_1 as D_{base} for alleviating users' burden of specifying a tolerance. L_1 causes the time warping distance highly affected by the lengths of the time-warped sequences; thus 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 the search performance. Since the time warping distance requires large CPU cost, we need to discard non-qualifying sequences as early as possible. In case of L_1, such decisions happen at the time the distance accumulated for a while exceeds a tolerance. In 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 CPU cost much less than that with L_1.

Note, however, that the index-based approach proposed in this paper is applicable to the case employing L_1 or L_2 as D_{base} in the same way.

4.2 Indexing Strategy

The goal of our method is to guarantee **no false dismissal** and **high search performance**. Most of indexes implicitly or explicitly assume that their distance function satisfies the triangular inequality. When employing a distance function not satisfying the triangular inequality, they cause false dismissal in searching. Yi et al. [25] 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, search performance without any index seriously degrades in large database environment.

For resolving the problem, we devise a new distance function D_{tw-lb} that consistently lower-bounds D_{tw} and also satisfies the triangular inequality. For 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 have to 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 3: Given two sequences S and Q, the lower-bound distance function $D_{tw-lb}(S, Q)$ is defined as follows:

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

■

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.

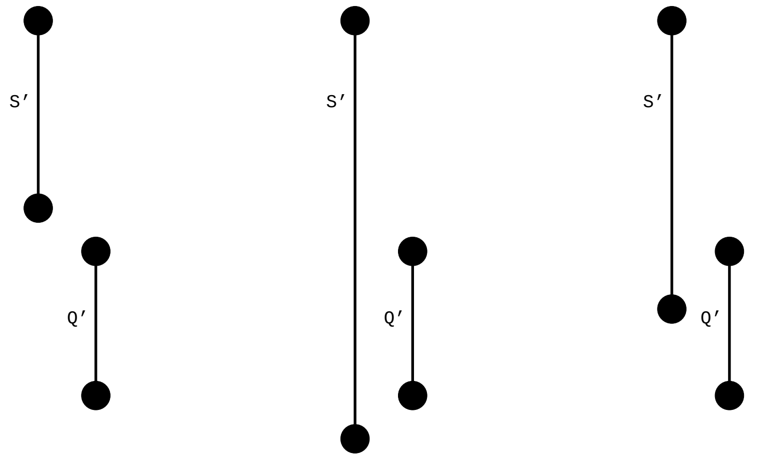
$$D_{tw}(S,Q) \;\geq\; D_\infty(\langle First(S), Last(S) \rangle, \langle First(Q), last(Q) \rangle) \tag{4.1}$$

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

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

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 easier explanation, let us assume that $Greatest(S') \geq Greatest(Q')$. If this assumption does not valid, the roles of S' and Q' can be exchanged. Let us consider each arrangement of ranges.



(a) S′ and Q′ are disjoint. (b) S′ encloses Q′. (c) S′ and Q′ overlap.

Figure 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.

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

■

Proof: Since $D_{tw-lb}(X,Z) = L_{\infty}(Feature(X), Feature(Z))$ and the distance function L_{∞} is a metric that always satisfies the triangular inequality [19], 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 to a point in 4-dimensional space since a 4-tuple feature vector is extracted from a sequence for indexing. For indexing a set of 4-dimensional points, any multi-dimensional indexes such as the R-tree [13], R^{+}-tree [22], R^{*}-tree [3], and X-tree [5] 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 gains in construction by hiring bulk loading methods [6, 14, 15].

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, 10]. 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.

Algorithm 1: Query Processing Algorithm TW-Sim-Search

Input : query sequence Q, tolerance ϵ
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 ϵ ;
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 environment for experiments and Section 5.2 shows and analyzes experimental results.

5.1 Experiment Environment

Two kinds of data sets were used for experiments: synthetic data set and stock data set. 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.

For each experiment, we performed 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 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.

For performance evaluation, we compared TW-Sim-Search, our method, with the previous ones: Naive-Scan [4], LB-Scan [25], and ST-Filter [18]. For fare evaluation, we slightly modified all the previous methods such that they employed L_∞ instead of L_1 as D_{base} for the time warping distance.[3] For the best performance of ST-Filter, we generated 100 categories using the equal-length-interval method [18]. We excluded the FastMap method [25] in performance evaluation since it has deficiency of causing false dismissal. For TW-Sim-Search, we employed the R-tree [13] as a multi-dimensional index, and set its page size to 1KB.

5.2 Results and Analysis

In Experiment 1, the candidate ratio of all the methods was compared using the S&P 500 stock data set. The candidate ratio is defined as follows:

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

Note that the number of data sequences was fixed at 545. The number of candidate sequences is different according to the method used to perform 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 step is not required in Naive-Scan, data sequences contained in the final query result are depicted in the

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

[3]We also performed the identical experiments with L_1 as D_{base}. As mentioned in Section 4.1, the overall performance of all the four methods became worse than that with L_∞ due to the CPU overhead with L_1. However, the trends appeared to be similar to that with L_∞. Due to space limitations, we omit these results in this paper.

graph as candidate sequences. The purpose of this experiment was to compare the filtering effects of the four methods by examining the trend of false alarm [1, 10] in each method.

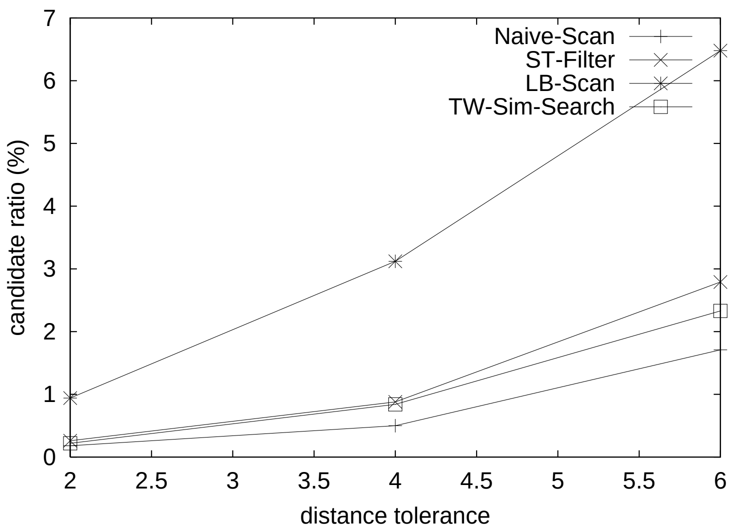


Figure 2: Comparison of filtering effects using S&P 500 stock data sequences.

Figure 2 depicts the results showing the candidate ratio of the four methods. The X-axis represents the tolerance and the Y-axis the candidate ratio. Since Naive-Scan does not produce false alarm, the number of candidate sequences is identical to the number of sequences in the final query result. As a method has the candidate ratio closer to that of Naive-Scan, we evaluate its filtering effect better. 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 the filtering effect slightly better than ST-Filter that is much better than LB-Scan.

The good filtering effect does not always result in good performance since the time spent in the filtering stage is different depending on the methods to be 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 the elapsed time. ST-Filter showed 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.

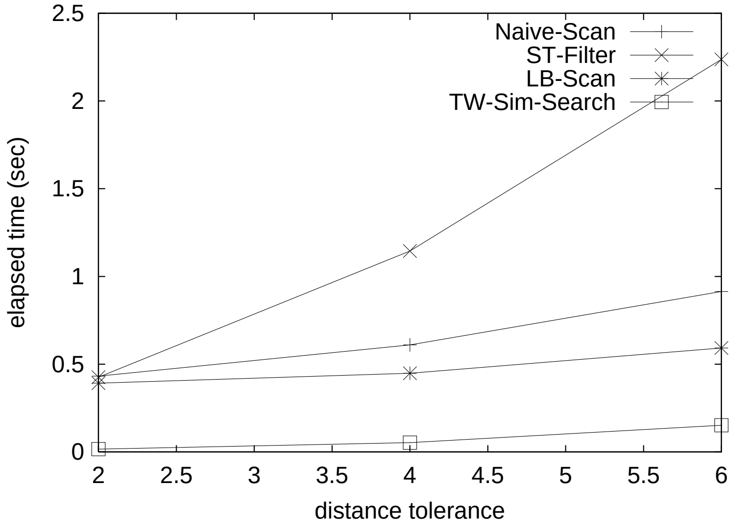


Figure 3: Comparison of elapsed times using S&P 500 stock data sequences.

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 4 times to 43 times speedup 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 gets larger as a tolerance decreases. This property is quite desirable because most users are interested in just a few number of answers.

The S&P 500 with only 545 data sequences are 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 synthetic data set.

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 of 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 all the database (in cases of Naive-Scan and LB-Scan) or traverse the large suffix tree. It is very

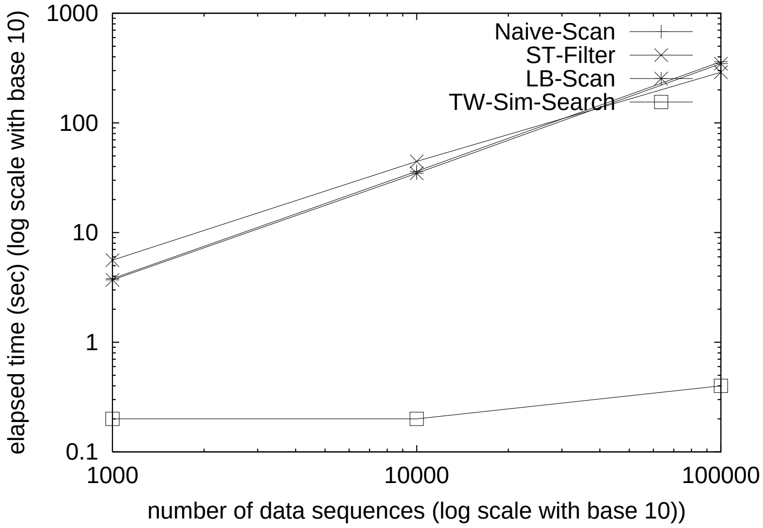


Figure 4: Comparison of elapsed times using synthetic data with different numbers of sequences.

interesting that ST-Filter performs worse than Naive-Scan and LB-Scan with 1,000 and 10,000 data sequences, however, 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 gain increases 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.

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.

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 unchanged relatively. 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 gets larger with the increasing length of data sequences.

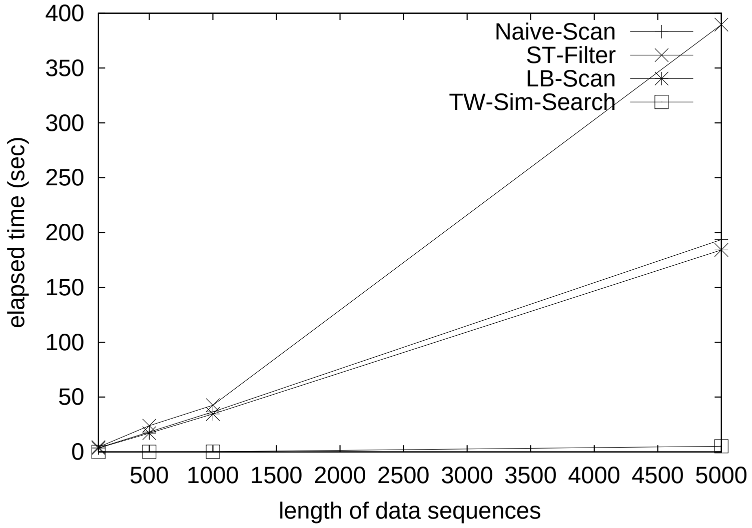


Figure 5: Comparison of elapsed times using synthetic data sequences with different lengths.

6 Concluding Remarks

Similarity Search is an operation that finds data sequences whose changing patterns are similar to that of a query sequence [1, 11], and is of growing importance in such applications as data mining and data warehousing [7, 21]. 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 [18, 25]. 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 [25]. 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 [25]. Berndt et al. [4] and Yi et al. [25] proposed methods that reduce the CPU cost. However, they have to scan all the database, thus are not suitable for large database environment. Park et al. [18] suggested a method that hires 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. [25] also proposed another method that highly accelerates the search performance. However, this method suffers from a critical problem that it could 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 is 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 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 the large synthetic data set. The performance gain becomes larger: (1) as the number of data sequences gets larger, (2) the average length of data sequences gets longer, and (3) as the tolerance in a query gets smaller. 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 without any trouble. 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.

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] N. Beckmann, H. Kriegel, R. Schneider, B. Seeger, "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles", *Proc. ACM SIGMOD*, pp. 322-331, 1990.

[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] S. Berchtold, D. A. Keim, H. Kriegel, "The X-tree: An Index Structure for High-Dimensional Data", *Proc. VLDB*, pp. 28-39, 1996.

[6] J. Bercken, B. Seeger, P. Widmayer, "A General Approach to Bulk Loading Multidimensional Index Structures", *Proc. VLDB*, pp. 406-415, 1997.

[7] 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.

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

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

[10] C. Faloutsos, K. Lin, "FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets", *Proc. ACM SIGMOD*, pp. 163-174, 1995.

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

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

[13] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching", *Proc. ACM SIGMOD*, pp. 47-57, 1984.

[14] I. Kamel, C. Faloutsos, "On Packing R-trees", *Proc. ACM CIKM*, pp. 490-499, 1993.

[15] S. T. Leutenegger, J. M. Edgington, M. A. Lopez, "STR: A Simple and Efficient Algorithm for R-tree Packing", *IEEE ICDE*, pp. 497-506, 1997.

[16] 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", submitted for publication.

[17] 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*, 2000. (accepted to appear)

[18] 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.

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

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

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

[22] T. K. Sellis, N. Roussopoulos, C. Faloutsos, "The R$^+$-Tree: A Dynamic Index for Multi-Dimensional Objects", *Proc. VLDB*, pp. 507-518, 1987.

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

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

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