Prefix-Querying: An Approach for Effective Subsequence Matching Under Time Warping in Sequence Databases

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

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

June-Suh Cho
IBM T.J. Watson Research Center
junesuh@us.ibm.com

Sriram Padmanabhan
IBM T.J. Watson Research Center
srp@us.ibm.com

ABSTRACT

This paper discusses an index-based *subsequence* matching that supports time warping in large sequence databases. Time warping enables finding sequences with similar patterns even when they are of different lengths. In our earlier work, we suggested an efficient method for *whole* matching under time warping. This method constructs a multi-dimensional index on a set of feature vectors, which are invariant to time warping, from data sequences. For filtering at feature space, it also applies a lower-bound function, which consistently underestimates the time warping distance as well as satisfies the triangular inequality.

In this paper, we incorporate the *prefix-querying approach* based on sliding windows into the earlier approach. For indexing, we extract a feature vector from every subsequence inside a sliding window and construct a multi-dimensional index using a feature vector as indexing attributes. For query processing, we perform a series of index searches using the feature vectors of *qualifying* query prefixes. Our approach provides effective and scalable subsequence matching even with a large volume of a database. We also prove that our approach 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 with real-world S&P 500 stock data and with very large synthetic data.

Keywords

Similarity search, Sequence database, Subsequence matching, Time warping, Indexing technique, Prefix querying

1. INTRODUCTION

The sequence database is a set of data 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, 9]. Similarity search is an operation that finds sequences or subsequences whose changing patterns are similar to that of a given query sequence [1, 2, 9]. Similarity search is of growing importance in many new applications such as data mining and data warehousing [6, 17].

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, 7, 9, 10, 17] map the sequences into points in a k-dimensional space ($k \leq n$) 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, 7], shifting [2, 7], normalization [8, 10], moving average [17], and time warping [5, 13, 14, 21].

Time warping is a transformation that allows any sequence element to replicate itself as many times as needed without extra costs [21]. 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 time warping distance fits well with the databases where sequences are of different lengths. Kim et al. [13] enumerated the situations where sequences of different lengths need to be compared.

For efficient processing of similarity search, most previous approaches [1, 2, 9] employ multi-dimensional indexes [3, 4, 18]. Yi et al. [21] claimed that the multi-dimensional indexes assuming the triangular inequality [15] directly or indirectly cause *false dismissal* in similarity search when their distance

ACM Tenth International Conference on Information and Knowledge Management (CIKM 2001) Atlanta, Georgia USA, 2001

functions do not satisfy the triangular inequality. False dismissal is to miss part of the final query result. Yi et al. 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.

In our earlier work [13], we suggested an efficient *whole* matching approach which constructs a multi-dimensional index from a set of feature vectors invariant to time warping and applies a new lower-bound time warping distance function D_{tw-lb} at feature space. Since the distance function D_{tw-lb} consistently underestimates the time warping distance and also satisfies the triangular inequality, this method achieves a high performance in query processing without incurring false dismissal.

As an extension of our earlier work, this paper proposes an effective approach for *subsequence* matching which supports time warping without the occurrence of any false dismissal. We could take the straightforward approach: 1) scan the database and extract the feature vector from *every* possible subsequence in each data sequence, 2) construct a multi-dimensional index from a set of all feature vectors, and 3) apply D_{tw-lb} as a distance function at feature space. This approach is feasible but not suitable for large sequence databases because the number of feature vectors to be stored in a multi-dimensional index is *quadratic*.

To provide a scalable subsequence matching, this paper proposes the *prefix-querying approach* based on sliding windows. Our approach is based on the following theorem: Given two sequences s and q, and a positive constant w ($1 \leq w \leq |s|$), if the time warping distance of s and q is within ε, then there is at least one prefix of q whose distance to $s[1:w]$ is within ε. Here, $|s|$ denotes the length of s and $s[1:w]$ denotes a prefix of s with length w.

For indexing, we place a sliding window of length w at every possible position (offset) on each data sequence and extract a feature vector from the subsequence corresponding to the window. Then, we construct a multi-dimensional index using a feature vector as indexing attributes. When a query sequence is submitted, we decide a set of *qualifying* query prefixes, extract a feature vector from each qualifying prefix, and perform a series of range queries on the multi-dimensional index using D_{tw-lb} as a distance function. Finally, we collect all the candidates returned by the range queries, leave out the duplicate candidates, and perform the post-processing to obtain the final answers.

To show the robustness of our approach, we prove that it does not incur false dismissal in query processing. To verify the superiority of our approach, we perform extensive experiments. The results reveal that our approach achieves significant speedup with real-world S&P 500 stock data and with 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 subsequence matching methods supporting time warping and points out their problems. Section 4 proposes the strategy and algorithms of our approach in detail. Section 5 evaluates the performance of the proposed approach through 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 numeric elements. $|S|$ is the length of S and $S[i]$ is the i-th element of S. $S[i:j]$ denotes the subsequence of S containing the elements from position i to j. $First(S)$ and $Last(S)$ are the first and the last elements of S. $Rest(S)$ is a subsequence of S that includes the elements from position 2 to the end. $\langle\rangle$ denotes an empty sequence.

L_p distance metric has been widely used to measure the similarity of two sequences s and q of the same length. L_1 is the Manhattan distance, L_2 is the Euclidean distance, and L_∞ is the maximum distance in any pair of elements [19].

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

Time warping is a generalization of classical algorithms for comparing discrete sequences with sequences of continuous values [16]. To find the minimum difference between two sequences, time warping enables each element of a sequence to match one or more neighboring elements of another sequence.

Definition 1. Given two sequences s and q, the time warping distance D_{tw} is defined recursively as follows [16]:

$$\begin{array}{lcl} D_{tw}(\langle\rangle, \langle\rangle) & = & 0 \\ D_{tw}(s, \langle\rangle) & = & D_{tw}(\langle\rangle, q) = \infty \\ D_{tw}(s,q) & = & D_{base}(First(s), First(q)) \ + \\ & & min \left\{ \begin{array}{l} D_{tw}(s, Rest(q)) \\ D_{tw}(Rest(s), q) \\ D_{tw}(Rest(s), Rest(q)) \end{array} \right. \end{array}$$

Here, D_{base} can be any L_p metric that returns the distance of two elements. The sequences whose time warping distances to a query sequence are not larger than a given tolerance are considered similar.

3. RELATED WORK

This section briefly summarizes the previous methods[1] for similarity search that supports time warping, and points out the possible problems when they are applied to subsequence matching.

3.1 Naive-Scan

There have been many research efforts on similarity search supporting time warping for voice recognition [16]. The method in this area reads all the data subsequences and computes D_{tw} between a data subsequence s and a query sequence q using the dynamic programming technique [5, 16].

This method incurs high CPU cost since 1) there are $O(mn^2)$ data subsequences in a set of m data sequences with average length n, and 2) the dynamic programming technique has the computation cost 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.

[1] Yi et al. [21] proposed the *FastMap* method using a multi-dimensional index. We do not include this method in our related work because it incurs false dismissal.

3.2 LB-Scan

Yi et al. [21] proposed the whole matching method supporting time warping. This method employs a distance function D_{lb} to discard non-qualifying sequences without false dismissal. D_{lb} saves CPU cost due to its reduced time complexity of $O(|s| + |q|)$.

This method can be extended for subsequence matching by computing the distance between a query sequence and every data subsequence using D_{lb}. 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 ST-Filter

Park et al. [14] proposed a method called ST-Filter that uses the suffix tree [20] as an index structure. 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 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. To make ST-Filter more effective, this method applies *categorization*. 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 cost and the suffix tree access cost depending on the granularity of categories.

3.4 LB-Filter

We suggested an index-based method [13] for *whole* matching. It employs a lower-bound distance function D_{tw-lb} to discard non-qualifying sequences quickly without false dismissal. D_{tw-lb} consistently underestimates the time warping distance and also satisfies the triangular inequality. Given two sequences s and q, D_{tw-lb} uses their feature vectors $Feature(s)$ and $Feature(q)$ as input parameters. The feature vector of a sequence s is represented as $Feature(s)$ = $(First(s)$, $Last(s)$, $Greatest(s)$, $Smallest(s))$. $First(s)$ 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. D_{tw-lb} is defined as follows:

Definition 2. Given two sequences s and q, the lower-bound distance function $D_{tw-lb}(s, q)$ is defined as follows [13]:

$$D_{tw-lb}(s,q) \;=\; L_p(Feature(s), Feature(q))$$

Here, L_p is defined as the same as D_{base} employed in D_{tw}.

This method extracts a feature vector from each data sequence and constructs a 4-dimensional index using a feature vector as indexing attributes. When a query sequence q is submitted with a distance tolerance ε, it first extracts a 4-tuple feature vector $Feature(q) = (First(q)$, $Last(q)$, $Greatest(q)$, $Smallest(q))$ and performs a square-range query on a 4-dimensional index. The range query uses a 4-tuple feature vector obtained from q as a central point and ε as a range for each dimension.

This method could be extended for subsequence matching using the following straightforward way: 1) scan the database and extract a feature vector from *every* possible subsequence in each data sequence, 2) construct a multi-dimensional index from a set of all feature vectors, and 3) extract a feature vector from a query sequence and perform a square-range query on the index using D_{tw-lb} as a distance function. This is feasible but not scalable because the number of feature vectors (4-dimensional data points) to be stored in an index is quadratic to the average length of data sequences. We call this method *LB-Filter* in this paper.

4. THE PROPOSED METHOD

We first define the problem we are going to solve. Given a set of m data sequences $S_1, S_2, ..., S_m$, a query sequence q, and a distance tolerance ε, we find a set of subsequences s whose time warping distances to q are not larger than ε.

This section proposes an efficient solution to the above problem. Our primary goal is to guarantee 1) **no false dismissal**, 2) **high search performance**, 3) **small storage overhead**. 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. Section 4.4 discusses the method to reduce the number of index searches occurred in processing of a query.

4.1 Similarity Model

For any two sequences, we measure their similarity using the time warping distance that employs L_∞ as D_{base}. For this similarity model, the time warping distance shown in Definition 1 is rewritten as in Definition 3.

Definition 3. Given two sequences s and q, their time warping distance D_{tw} is defined recursively as follows:

$$\begin{aligned}
D_{tw}(\langle\rangle,\langle\rangle) &= 0\\
D_{tw}(s,\langle\rangle) &= D_{tw}(\langle\rangle,q) = \infty\\
D_{tw}(s,q) &= max\left\{\begin{array}{l} |First(s) - First(q)| \\ min\left\{\begin{array}{l} D_{tw}(s, Rest(q)) \\ D_{tw}(Rest(s), q) \\ D_{tw}(Rest(s), Rest(q)) \end{array}\right. \end{array}\right.
\end{aligned}$$

Unlike the previous methods [5, 14, 21], 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. When we consider a subsequence matching, the situation becomes worse due to the various lengths of subsequences. Thus, it is very difficult for a user to decide a distance tolerance appropriate for the characteristics of a target database. Especially, in dynamic environment where data sequences keep growing, it is almost impossible to determine the optimal distance tolerance. By using L_∞, however, users can easily specify an appropriate tolerance without considering the lengths.

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[2].

[2]Theorem 1 and Corollary 1 also hold when we apply L_1 or

4.2 Indexing Strategy

To provide a scalable subsequence matching, this paper proposes the *prefix-querying approach* based sliding windows. The proposed approach is based on the following theorem.

THEOREM 1. *Given two sequences s and q, and a positive constant w ($1 \leq w \leq |s|$), if the time warping distance of s and q is within ε, then there is at least one query prefix $q[1:j]$ ($1 \leq j \leq |q|$) whose distance to $s[1:w]$ is within ε. That is,*

$$D_{tw}(s,q) \leq \varepsilon \implies \exists_j \ (D_{tw}(s[1:w], q[1:j]) \leq \varepsilon)$$

PROOF. Let $p = \langle p[1], p[2], ..., p[|p|]\rangle$ be the *warping path* [5] that generates the minimum difference between s and q ($|s| \leq |p|$, $|q| \leq |p|$). Each element of the warping path is represented as $p[h] = (s[i_h], q[j_h])$ ($1 \leq h \leq |p|$, $1 \leq i_h \leq |s|$, $1 \leq j_h \leq |q|$). Then, the distance of s and q is computed as:

$$D_{tw}(s,q) \ = \ max_{h=1}^{|p|} \ (|s[i_h] - q[j_h]|)$$

Let w be an arbitrary constant between 1 and $|s|$. Then, by the monotonic and continual property [5] of a warping path, there must be at least one element $p[h']$ in the warping path such that $p[h'] = (s[w], q[j_{h'}])$, $(1 \leq h' \leq |p|, 1 \leq j_{h'} \leq |q|)$. Since the following inequality always holds, the sub-warping path from $p[1]$ to $p[h']$ always generates the distance smaller than or equal to the distance of the warping path p.

$$max_{h=1}^{h'} \ (|s[i_h] - q[j_h]|) \ \leq \ max_{h=1}^{|p|} \ (|s[i_h] - q[j_h]|)$$

Therefore, if the warping path p has the distance not larger than ε, the sub-warping path from $p[1]$ through $p[h']$ also has the distance not larger than ε. Since the sub-warping path from $p[1]$ to $p[h']$ represents the distance between $s[1:w]$ and $q[1:j_{h'}]$ ($1 \leq j_{h'} \leq |q|$), we prove the theorem. □

Since the distance function D_{tw-lb} consistently underestimates D_{tw} [13], we derive the following corollary from Theorem 1.

COROLLARY 1. *Given two sequences s and q, and a positive constant w ($1 \leq w \leq |s|$), if $D_{tw}(s,q) \leq \varepsilon$, then there is at least one query prefix $q[1:j]$ ($1 \leq j \leq |q|$) which satisfies $D_{tw-lb}(s[1:w], q[1:j]) \leq \varepsilon$. That is,*

$$D_{tw}(s,q) \leq \varepsilon \implies \exists_j \ (D_{tw-lb}(s[1:w], q[1:j]) \leq \varepsilon)$$

To absorb the main idea of Corollary 1 into our indexing scheme, we utilize the *sliding window concept* [9]. That is, we place a window of size w at every possible position on each data sequence and extract a 4-tuple feature vector ($First(s)$, $Last(s)$, $Greatest(s)$, $Smallest(s)$) from the subsequence s identical to that window. Then, we construct a 4-dimensional index using a feature vector as indexing attributes. While the number of all possible subsequences in a data sequence of size n is $\frac{n(n+1)}{2}$, the number of subsequences stored in the index is $n - w + 1$. Therefore, the storage overhead with our indexing scheme keeps small even in a large database.

The search performance is closely related to the window size w. For a subsequence s and a query sequence q, as the window size w increases, the difference of $D_{tw}(s,q)$ and $D_{tw-lb}(s[1:w], q[1:j])$ ($1 \leq j \leq |q|$) also increases, and thus incurs more *false alarms* [9]. Moreover, the number of query prefixes to be compared with $s[1:w]$ increases as the difference of $|q|$ and w becomes larger. These two facts imply that the performance gain of our indexing scheme becomes smaller as $|q| - w$ increases. On the other hand, if $|q|$ is smaller than w, the 4-dimensional index on windows of size w is useless in subsequence matching.

To overcome these two problems, we propose to use the concepts of $maxWarpRatio$ [5] and $minQLen$. $minQLen$ represents the minimum query length, and $maxWarpRatio$ represents how many times each element of a sequence can replicate itself by time warping. If $maxWarpRatio$ is 3, for example, then a sequence s of length n can be transformed into a sequence s' by time warping where $n \leq |s'| \leq 3n$. Assuming that $minQLen$ and $maxWarpRatio$ are determined in advance according to the characteristics of a target application, we decide the size of a sliding window as $w = \lceil \frac{minQLen}{maxWarpRatio} \rceil$. w may be considered as the length of the shortest subsequence that can be matched to a query sequence of length $minQLen$.

Example 1. When $minQLen$ is 20 and $maxWarpRatio$ is 4, $w = \lceil \frac{minQLen}{maxWarpRatio} \rceil = \lceil \frac{20}{4} \rceil = 5$. Consider a data sequence $S = \langle 2,3,3,4,3,6,6,7,9,5\rangle$. To build an index, we first place a sliding window of size 5 at every possible position. The number of possible windows is $|S| - w + 1 = 10 - 5 + 1 = 6$. Then we extract a feature vector from each window and insert it into a 4-dimensional index. The windows and their corresponding feature vectors are shown in Table 1.

When a query sequence q is submitted, we first decide the minimum and the maximum lengths of query prefixes that can be matched to a data subsequence of size w as $minPLen = \lceil \frac{w}{maxWarpRatio} \rceil$ and $maxPLen = w * maxWarpRatio$. Then, for each query prefix $q[1:j]$ ($minPLen \leq j \leq maxPLen$), we compute a 4-tuple feature vector $Feature(q[1:j])$ and perform a square-range query on the 4-dimensional index using D_{tw-lb} as a distance function. Finally, we collect all the candidates returned by index searches, leave out the duplicate candidates, and perform the post-processing to obtain the final answers.

Example 2. Let $minQLen$, $maxWarpRatio$, and w be 20, 4, and 5 respectively. Then $minPLen = \lceil \frac{w}{maxWarpRatio} \rceil = \lceil \frac{5}{4} \rceil = 2$, and $maxPLen = w * maxWarpRatio = 5*4 = 20$. For a query sequence q whose length is at least $minQLen$ ($= 20$), the qualifying prefixes are $q[1:j]$ ($2 \leq j \leq 20$).

4.3 Algorithms

This subsection presents the algorithms for index construction and query processing used in our approach for subsequence matching.

4.3.1 Index Construction

Each sliding window in data sequences is mapped to a point in 4-dimensional space since a 4-tuple feature vector is extracted from a subsequence corresponding to that window. For indexing a set of 4-dimensional points, any multi-dimensional indexes such as the R-tree [11], R*-tree [3], and X-tree [4] can be used. The index construction algorithm first makes an entry ($First(w)$, $Last(w)$, $Greatest(w)$,

L_2 to D_{tw} and D_{tw-lb} as D_{base}. Thus, when applications prefer L_1 or L_2 over L_∞, we just need to replace L_∞ with L_1 or L_2 in both courses of index search and post-processing.

Window	First	Last	Greatest	Smallest	Feature Vector
$S[1:5] = \langle 2,3,3,4,3 \rangle$	2	3	4	2	$(2,3,4,2)$
$S[2:6] = \langle 3,3,4,3,6 \rangle$	3	6	6	3	$(3,6,6,3)$
$S[3:7] = \langle 3,4,3,6,6 \rangle$	3	6	6	3	$(3,6,6,3)$
$S[4:8] = \langle 4,3,6,6,7 \rangle$	4	7	7	3	$(4,7,7,3)$
$S[5:9] = \langle 3,6,6,7,9 \rangle$	3	9	9	3	$(3,9,9,3)$
$S[6:10] = \langle 6,6,7,9,5 \rangle$	6	5	9	5	$(6,5,9,5)$

Table 1: Windows and their feature vectors from a data sequence $S = \langle 2,3,3,4,3,6,6,7,9,5 \rangle$ when the window size is 5.

$Smallest(w)$, $ID(w)$) for each window w and then inserts it into a 4-dimensional index. Here, $ID(w)$, the identifier of w, consists of the identifier of a sequence S containing w and the starting offset of w inside S. If there are a large number of data sequences at the initial stage, we can highly reduce index construction time by hiring bulk loading methods [12].

4.3.2 Query Processing

Algorithm 1 shows TW-Subseq-Matching1, our query processing algorithm. We perform the index search for each the prefix of q whose length is between $minPLen$ and $maxPLen$.

Step-1 extracts a 4-tuple feature vector (F, L, G, S) from a prefix $q[1:j]$. Step-2 performs a square-range query on a 4-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 uses D_{tw-lb} for filtering. That is, Step-2 finds all the data points contained in $([F-\varepsilon, F+\varepsilon], [L-\varepsilon, L+\varepsilon], [G-\varepsilon, G+\varepsilon], [S-\varepsilon, S+\varepsilon])$. Step-3 merges a set of candidates retrieved in Step-2 into CandidateSet, during which duplicate candidates are removed.

Step-4, Step-5, and Step-6 perform the post-processing to discard *false alarms*. For each entry e in the candidate set, Step-4 retrieves the longest one s in a set of subsequences containing e as their prefix. Step-5 builds the *cumulative distance table* [5] between q and s, during which we apply the pruning method [14] to save the computation cost. Step-6 finds the prefixes of s whose actual distances to q are within ε, and inserts them into $answerSet$.

Input : query sequence q, tolerance ε
Output: set of answers $answerSet$

candidateSet := {};
answerSet := {};

for *each j between $minPLen$ and $maxPLen$* **do**
1 Extract a feature vector (F, L, G, S) from $q[1:j]$;
2 queryResult := PerformRangeQuery $((F, L, G, S), \varepsilon)$;
3 candidateSet := candidateSet $\cup$ queryResult;

for *each entry e in candidateSet* **do**
4 Retrieve a corresponding subsequence s from DB;
5 Build a cumulative distance table T between q and s;
6 answerSet := answerSet $\cup$ FindAnswer(T);

return $answerSet$;

Algorithm 1: Query processing algorithm TW-Subseq-Matching1.

4.4 Reducing the Number of Index Searches

Our query processing algorithm traverses the index for each query prefix $q[1:j]$ $(minQLen \leq j \leq maxQLen)$. Therefore, the number of index searches is $maxPLen - minPLen + 1$. By the definitions of $minPLen$ and $maxPLen$, the number of index searches grows as the $maxWarpRatio$ increases. To maintain good query performance even with large $maxWarpRatio$, we need to reduce the number of index searches at the cost of filtering extra candidates. This subsection describes our strategy to reduce the number of index searches.

Instead of building a query rectangle for each query prefix, we take the approach of constructing a single query rectangle which contains multiple points or feature vectors of query prefixes. Let (F, L, G, S) and (F', L', G', S') be feature vectors from query prefixes $q[1:j]$ and $q[1:j']$, respectively $(minPLen \leq j, j' \leq maxPLen)$. With a distance tolerance ε, the query rectangles are generated using feature vectors as central points and ε as a range for each dimension. That is, the query rectangle for $q[1:j]$ is $([F-\varepsilon, F+\varepsilon]$, $[L-\varepsilon, L+\varepsilon]$, $[G-\varepsilon, G+\varepsilon]$, $[S-\varepsilon, S+\varepsilon])$ and the query rectangle for $q[1:j']$ is $([F'-\varepsilon, F'+\varepsilon]$, $[L'-\varepsilon, L'+\varepsilon]$, $[G'-\varepsilon, G'+\varepsilon]$, $[S'-\varepsilon, S'+\varepsilon])$. To merge these two query rectangles into a single query rectangle, we do the following: 1) compute the minimum bounding rectangle of the two 4-dimensional points (F, L, G, S) and (F', L', G', S'), and 2) extend each dimension using ε. The minimum bounding rectangle of (F, L, G, S) and (F', L', G', S') is represented as $([min(F, F'), max(F, F')], [min(L, L'), max(L, L')], [min(G, G'), max(G, G')], [min(S, S'), max(S, S')])$. Then, the new query rectangle is represented as $([min(F, F') - \varepsilon, max(F, F') + \varepsilon], [min(L, L') - \varepsilon, max(L, L') + \varepsilon], [min(G, G') - \varepsilon, max(G, G') + \varepsilon], [min(S, S') - \varepsilon, max(S, S') + \varepsilon])$.

This process can be generalized to merge more than two query rectangles. At the final stage, all the query rectangles could be merged into a single query rectangle, which significantly reduces the cost for index searching but also introduces the cost for extra candidates. As the number of extra candidates increases, the cost for post-processing also grows. Therefore, we need a strategy which constructs optimal query rectangles considering both index searching and post-processing costs. We can think of the following three strategies.

- **Strategy 1:** construct a query rectangle for each query prefix.
- **Strategy 2:** make a large query rectangle containing all the points corresponding to all the query prefixes.
- **Strategy 3:** divide query prefixes into multiple groups

and construct a query rectangle for each group[3].

In general, if there are a large number of prefixes, it is more effective to build multiple query rectangles since it prevents each query rectangle from becoming too large. However, the prefixes from the same query sequence have common values for the first, greatest, and smallest elements. Thus, their feature vectors tend to be located closely in feature space. This tendency keeps the query rectangle from getting abnormally large even when we use only a single query rectangle for all the prefixes. Since finding the optimal number of query rectangles is not a focus of this paper, we employ the first two strategies in performance evaluation to simplify the problem, and defer the detailed discussion on the third as a further study. We call the subsequence matching algorithm based on the second strategy as TW-Subseq-Matching2.

5. PERFORMANCE EVALUATION

This section presents the experimental results for performance evaluation of the proposed approach. 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 and real-life stock data sets. 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 $S[1]$ was 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) extract a random subsequence from that data sequence; (3) take a random value from an appropriate range[4] for every element of the subsequence; and (4) add the value to its corresponding element. We use an average elapsed time of processing 100 queries as a performance measure.

The hardware platform for the experiments is the IBM RS/6000 system equipped with 256 MB RAM and 4.5 GB hard disk with a data rate of 10 MB/sec approximately. The software platform is the AIX version 4.3, which was set to a single-user mode to minimize the interference from other system and user processes.

For performance evaluation, we compared our two methods TW-Subseq-Matching1 and TW-Subseq-Matching2 with LB-Scan [21] and ST-Filter [14]. For fair evaluation, we slightly modified the previous methods such that they employed L_∞ instead of L_1 as D_{base} for the time warping distance. For ST-Filter, we generated 50 categories using the *maximum entropy method* [14]. For TW-Subseq-Matching1 and TW-Subseq-Matching2, we employed the R-tree [11] as a multi-dimensional index, and set its page size to 1KB. Table 2 summarizes the parameters used in each experiment.

5.2 Results and Analyses

Experiment 1 compares the elapsed times of the four methods with increasing distance tolerance ε from 0.2 to 1.2[5]. We set the values of other parameters $maxWarpRatio$, $minQLen$, and $avgQLen$ as 5, 50, and 90, respectively. Here, $avgQLen$ represents the average length of query sequences actually used in the experiments. Figure 1 shows the experimental result with S&P 500 data sequences. Note that in all the figures, the Y-axis is in the log scale with base-10. According to this result, all the methods show increasing elapsed times as ε becomes larger. Let us compare LB-Scan and ST-Filter first. ST-Filter shows better performance when ε is small (0.2, 0.4, and 0.6), but worse performance when ε is large (0.8, 1.0, and 1.2). This result is mainly due to the large volume of the suffix tree. Actually, the size of a suffix tree is much larger than that of its database [13]. When ε is small, ST-Filter accesses just a small portion of the suffix tree, and thus shows good performance. When ε gets large, ST-Filter touches a large portion of the huge suffix tree in searching, and thus its performance gain lessens. The result also reveals that both of our methods significantly outperform the previous two methods. Let us now compare TW-Subseq-Matching1 with TW-Subseq-Matching2. For small values (0.2, 0.4, and 0.6) of ε, TW-Subseq-Matching1 shows better performance than TW-Subseq-Matching2. However, TW-Subseq-Matching2 is the winner when ε is large (0.8, 1.0, and 1.2). This is because, with large ε, the single query rectangle introduces relatively a smaller number of additional false alarms and also enjoys the low cost of just one index search. TW-Subseq-Matching1 is shown to be approximately 42 times faster than the best previous method (ST-Filter) when using ε of 0.2, and TW-Subseq-Matching2 is approximately 6 times faster than the best previous method (LB-Scan) when using ε of 1.2.

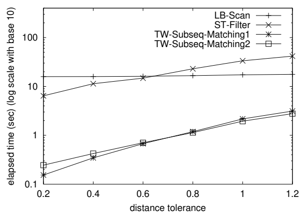

Figure 1: Comparison of elapsed times with increasing distance tolerance ε (S&P 500 data set).

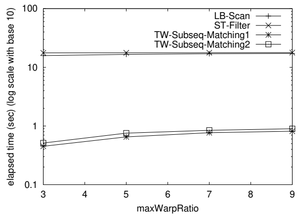

Figure 2: Comparison of elapsed times with increasing $maxWarpRatio$ (S&P 500 data set).

Experiment 2 examines the performance of the four methods with changing $maxWarpRatio$ of 3, 5, 7, and 9. We set other parameters ε, $minQLen$, and $avgQLen$ as 0.6, 50, and 90, respectively. Figure 2 shows the result with

[3]We are considering to use the l-adaptive method [9] for this scheme as future work.

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

[5]The average number of answers was 8 when $\varepsilon = 0.2$ and 1703 when $\varepsilon = 1.2$.

	exp 1	exp 2	exp 3	exp 4	exp 5	exp 6
data set	S&P 500	S&P 500	S&P 500	S&P 500	synthetic	synthetic
seqNum	545	545	545	545	1000 - 4000	500
seqLen	231	231	231	231	200	200 - 800
avgQLen	90	90	90	50 - 110	90	70 - 280
minQLen	50	50	30 - 90	50	50	50 - 200
maxWarpRatio	5	3 - 9	5	5	5	5
window size	10	6 - 17	6 - 18	10	10	10 - 40
ε	0.2 - 1.2	0.6	0.6	0.6	0.1	0.1

Table 2: Parameter values for each experiment.

S&P 500 data sequences. The result shows slight increase of the elapsed time with increasing $maxWarpRatio$ in all the methods. This is because larger $maxWarpRatio$ generates more candidates as well as more answers. Compared with the best previous one, TW-Subseq-Matching1, which is shown to be better than TW-Subseq-Matching2 in all values of $maxWarpRatio$, achieves about 35 times speedup when $maxWarpRatio$ is 3, and about 21 times speedup when $maxWarpRatio$ is 9.

Experiment 3 compares the performance of the four methods with changing $minQLen$ of 30, 50, 70, and 90. We set other parameters ε, $maxWarpRatio$, and $avgQLen$ as 0.6, 5, and 90, respectively. Figure 3 shows the result with S&P 500 data sequences. While the performance of the previous methods is nearly constant regardless of $minQLen$, that of our methods gets better with increasing $minQLen$. This is because larger $minQLen$ tends to reduce the number of query prefixes, thus decreases the number of index searches in TW-Subseq-Matching1 or the number of false alarms in TW-Subseq-Matching2. Compared with the best previous one, TW-Subseq-Matching1 is shown to be about 23 times faster when using $minQLen$ of 30, and 28 times faster when using $minQLen$ of 90.

longer query sequence generates fewer answers. The result shows that TW-Subseq-Matching1 performs about 25 times better than the best previous method. Since the S&P 500 stock data with only 545 data sequences is too small to perform the scalability test, we employ a large volume of synthetic data sets to compare the performance of our methods with LB-Scan, which shows better performance than ST-Filter in large databases.

In Experiment 5, we increase the number of data sequences from 1,000 to 4,000 while fixing the other parameters ε, $maxWarpRatio$, $minQLen$, and $avgQLen$, at 0.1, 5, 50, and 90, respectively. We set the average length of data sequences as 200. Figure 5 shows the result with this synthetic data set. The elapsed time of the three methods increases linearly with the increasing number of sequences because 1) the performance is affected only by the number of retrieved answers when the window size, ε, and $maxWarpRatio$ are fixed, and 2) the number of answers increases linearly as the number of sequences grows. The result also shows that TW-Subseq-Matching1 performs about 19 times better with 1,000 sequences and 13 times better with 4,000 sequences compared with LB-Scan.

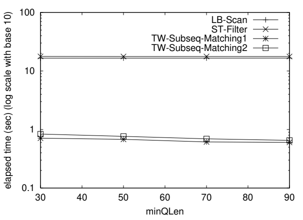

Figure 3: Comparison of elapsed times with increasing $minQLen$ **(S&P 500 data set).**

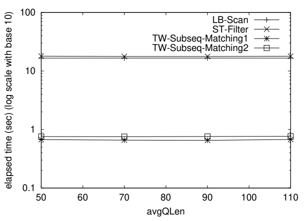

Figure 4: Comparison of elapsed times with increasing $avgQLen$ **(S&P 500 data set).**

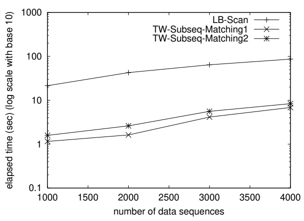

Figure 5: Comparison of elapsed times with increasing number of data sequences (synthetic data set).

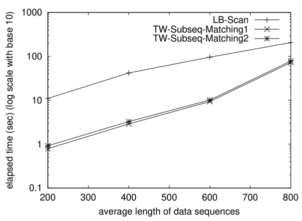

Figure 6: Comparison of elapsed times with increasing length of data sequences (synthetic data set).

Experiment 4 is to examine the performance of four methods with changing $avgQLen$ of 50, 70, 90, and 110. We set other parameters ε, $maxWarpRatio$, and $minQLen$ as 0.6, 5, and 50, respectively. Figure 4 shows the result with S&P 500 data sequences. According to the result, all the methods show almost constant performance independently of the values of $avgQLen$. This can be interpreted as: 1) even though the longer query sequence generally increases the computation cost, the increase is marginal when L_∞ is used as base distance function, and 2) when $maxWarpRatio$ is fixed, the

Experiment 6 changes the average length of data sequences from 200 to 800 while fixing the number of data sequences, ε, and $maxWarpRatio$ at 500, 0.1, and 5. We set $minQLen$ as 1/4 of the average length of data sequences. We also set $avgQLen$ as $minQLen + minQLen * (2/5)$. As shown in Figure 6, the performance tendency is quite similar to that in Figure 5, and our approach is also effective even with databases of long sequences.

6. CONCLUDING REMARKS

Similarity search is an operation that finds data sequences or subsequences whose changing patterns are similar to that of a query sequence, and is of growing importance in such applications as data mining and data warehousing. 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. In this paper, we have discussed efficient subsequence matching that supports the time warping transformation.

The time warping distance does not satisfy the triangular inequality. 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. In our earlier work, we suggested an approach for efficient *whole* matching which constructs a multi-dimensional index from a set of feature vectors invariant to time warping and applies a new distance function D_{tw-lb} at feature space. As an extension of our earlier work, this paper has proposed an index-based approach for *subsequence* matching which supports time warping without the occurrence of any false dismissal. Instead of taking the brute-force extension, we have hired the prefix-querying approach based on sliding windows in order to provide scalable subsequence matching with small storage overhead. For indexing, we extract a feature vector from every subsequence inside a sliding window and construct a multi-dimensional index using the feature vector as indexing attributes. For query processing, we perform a series of index searches using the feature vectors of *qualifying* query prefixes. We also have discussed the strategy to reduce the number of index searches occurred in processing of a query.

For performance evaluation, we compared our methods with previous ones through extensive experiments using synthetic and real-life data sets. The results show that 1) when a distance tolerance is small, TW-Subseq-Matching1 which constructs a query rectangle for each qualifying query prefix is better than TW-Subseq-Matching2 which makes a large query rectangle for all qualifying query prefixes, 2) when a distance tolerance is large, TW-Subseq-Matching2 has better performance than TW-Subseq-Matching1, 3) our proposed methods can be a few orders of magnitude faster than the previous best one, 4) the performance gain of our methods is still maintained with various values of $minQLen$, $avgQLen$ and $maxWarpRatio$, and 5) our methods are effective even with a large volume of databases.

7. ACKNOWLEDGMENTS

In performing this research, Sang-Wook Kim was supported by Korea Research Foundation with Grant KRF-2000-041-E00258. Sang-Wook Kim would like to thank Jung-Hee Seo, Suk-Yeon Hwang, Grace(Ju-Young) Kim, and Ju-Sung Kim for their encouragement and support.

8. REFERENCES

[1] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In *Proc. FODO*, pages 69–84, 1993.

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

[3] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proc. ACM SIGMOD*, pages 322–331, 1990.

[4] S. Berchtold, D. A. Keim, and H. Kriegel. The X-tree: An index structure for high-dimensional data. In *Proc. VLDB*, pages 28–39, 1996.

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

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

[7] K. W. Chu and M. H. Wong. Fast time-series searching with scaling and shifting. In *Proc. ACM PODS*, pages 237–248, 1999.

[8] G. Das, D. Gunopulos, and H. Mannila. Finding similar time series. In *Proc. PKDD*, pages 88–100, 1997.

[9] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *Proc. ACM SIGMOD*, pages 419–429, 1994.

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

[11] A. Guttman. R-tree: A dynamic index structure for spatial searching. In *Proc. ACM SIGMOD*, pages 47–57, 1984.

[12] I. Kamel and C. Faloutsos. On packing R-trees. In *Proc. ACM CIKM*, pages 490–499, 1993.

[13] S. W. Kim, S. Park, and W. W. Chu. An index-based approach for similarity search supporting time warping in large sequence databases. In *Proc. IEEE ICDE*, pages 607–614, 2001.

[14] S. Park, W. W. Chu, J. Yoon, and C. Hsu. Efficient searches for similar subsequences of different lengths in sequence databases. In *Proc. IEEE ICDE*, pages 23–32, 2000.

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

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

[17] D. Rafiei and A. Mendelzon. Similarity-based queries for time series data. In *Proc. ACM SIGMOD*, pages 13–24, 1997.

[18] T. K. Sellis, N. Roussopoulos, and C. Faloutsos. The R^+-tree: A dynamic index for multi-dimensional objects. In *Proc. VLDB*, pages 507–518, 1987.

[19] K. Shim, R. Srikant, and R. Agrawal. High-dimensional similarity joins. In *Proc. ICDE*, pages 301–311, 1997.

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

[21] B.-K. Yi, H. V. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. In *IEEE ICDE*, pages 201–208, 1998.