Index Interpolation: An Approach to Subsequence Matching Supporting Normalization Transform in Time-Series Databases*

Woong-Kee Loh†, Sang-Wook Kim‡, and Kyu-Young Whang†
†Department of Electrical Engineering & Computer Science, Division of Computer Science
Advanced Information Technology Research Center (AITrc)
Korea Advanced Institute of Science and Technology (KAIST)
‡Division of Computer, Information, and Communications, Kangwon National University
E-mail: {woong,kywhang}@mozart.kaist.ac.kr, wook@cc.kangwon.ac.kr

Abstract

In this paper, we propose a subsequence matching algorithm that supports normalization transform in time-series databases. Normalization transform enables finding sequences with similar fluctuation patterns although they are not close to each other before the normalization transform. Application of the existing whole matching algorithm supporting normalization transform to the subsequence matching is feasible, but requires an index for every possible length of the query sequence causing serious overhead on both storage space and update time. The proposed algorithm generates indexes only for a small number of different lengths of query sequences. For subsequence matching it selects the most appropriate index among them. We can obtain better search performance by using more indexes. We call our approach *index interpolation*. We formally prove that the proposed algorithm does not cause false dismissal. For performance evaluation, we have conducted experiments using the indexes for only five different lengths out of the lengths $256 \sim 512$ of the query sequence. The results show that the proposed algorithm outperforms the sequential scan by up to 14.6 times on the average when the selectivity of the query is 10^{-5}.

1 Introduction

Time-series data are sequences of real numbers sampled at a fixed time interval [6]. The examples are stock prices, exchange rates, weather data, product sales data, and medical measurements [2, 9]. Finding similar time-series data is one of the most challenging problems in the new database research areas such as data mining and data warehousing [2, 3]. Examples of such a problem are finding stock items with similar trends in prices, finding periods with similar temperature patterns, and finding products with similar sales trends [9, 15]. The time-series data stored in a database are called *data sequences*, and finding data sequences similar to a given query sequence from the database is called *similar sequence matching* [1, 2, 3, 9, 15].

Similar sequence matching algorithms are classified into whole matching and subsequence matching algorithms [9]. Whole matching finds data sequences that are similar to a query sequence, where the lengths of data sequences and the query sequence are all identical. Subsequence matching finds subsequences, contained in data sequences, that are similar to a query sequence of an arbitrary length. In general, subsequence matching is applicable to a wider range of applications than whole matching.

Existing similar sequence matching algorithms map a data sequence of length n to a point in an n-dimensional space. Most of the algorithms define similarity between two data sequences using the Euclidean distance between the two corresponding points [1, 7, 9, 10, 15, 18], although some use different similarity measures [2]. They use multidimensional index structures such as the R-tree [12], R$^+$-tree [16], and R*-tree [4] to efficiently store and retrieve n-dimensional points. Since the search performance degrades exponentially as the dimensionality of the index structures increases [5, 17], most of the existing algorithms reduce the dimensionality by mapping n-dimensional points into f-dimensional ones ($f < n$). Mapping functions such as the Discrete Fourier Transform (DFT) [14], Discrete Cosine Transform (DCT) [14], and Haar Wavelet Transform [11] are used to reduce dimensionality [1, 7, 9, 10, 15].

In this paper, we propose an efficient subsequence matching algorithm that supports normalization transform [10, 15]. The proposed algorithm enables finding a data sequence that has a fluctuation pattern similar

* This work has been supported by Korea Science and Engineering Foundation (KOSEF) through Advanced Information Technology Research Center (AITrc).

CIKM 2000, McLean, VA USA
© ACM 2000 1-58113-320-0/00/11 . . .$5.00

to the query sequence even though they are not close to each other before the normalization transform. For example, it is useful to find a stock item whose price increase/decrease pattern is similar to the given stock item regardless of their absolute prices.

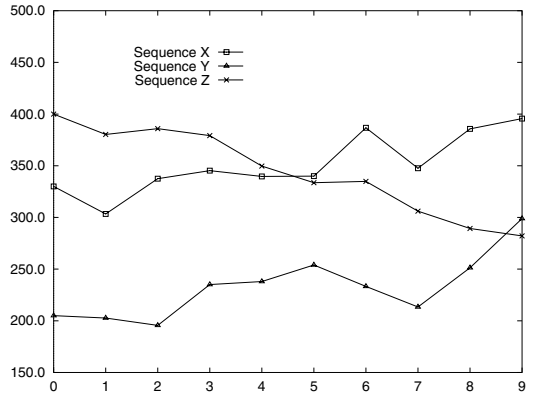
Existing algorithms supporting normalization transform handles only query sequences or windows of fixed lengths, while the proposed one does those of arbitrary lengths. To solve the problem, we may attempt simple application of the existing subsequence matching algorithms [2, 9]. However, it is not feasible since these algorithms do not have information for normalization transform of the arbitrary length subsequences. We explain the reason in detail in Section 3. We may also attempt application of the existing whole matching algorithm to support normalization transform [10]. However, since the whole matching algorithm supports query sequences of a fixed length, to support query sequences of arbitrary lengths, they must generate an index for each possible length of the query sequence. This would cause serious overhead on both storage space and time when inserting or deleting data sequences.

In this paper, we propose an efficient algorithm that overcomes those problems. The proposed algorithm generates indexes for only a few different lengths of the query sequences with some proper intervals and performs subsequence matching for every possible length of the query sequences. To show the correctness of the proposed algorithm, we prove that the algorithm does not cause false dismissal. We call this approach *index interpolation*. We can trade-off the search performance with storage space by adjusting the number of indexes. In this paper, we call the case with the indexes for the selected lengths of query sequences as the *selectively-indexed case* and the one for all the lengths as the *fully-indexed case*.

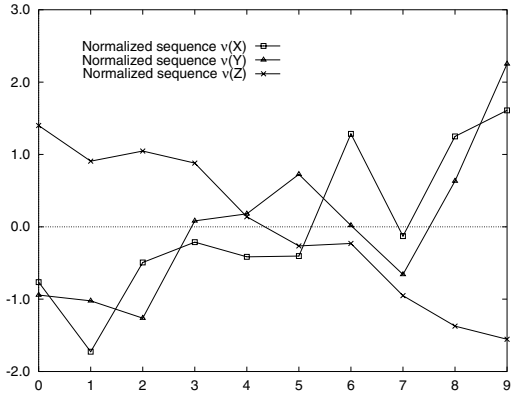
This paper is organized as follows. In Section 2, we formally define the problem of subsequence matching that supports normalization transform. In Section 3, we briefly introduce the existing subsequence matching algorithms [2, 9] and the whole matching algorithm that support normalization transform [10]. We also discuss the problems in applying them to normalization transform subsequence matching. In Section 4, we explain in detail the proposed indexing and searching algorithms. In Section 5, we evaluate the performance of the proposed algorithm by experiments. Finally, we summarize and conclude the paper in Section 6.

2 Problem Definition

In this section we formally define the normalization transform and the problem of subsequence matching that supports normalization transform. Table 1 summarizes the notation used in this paper.



(a) Original sequences.



(b) Normalized sequences.

Figure 1: An example of original and normalized sequences.

Definition 1 Given a sequence $\vec{X} = (x_i)$ $(0 \leq i < n)$ of length n (≥ 1), the *normalized sequence* $\nu(\vec{X}) = (\tilde{x}_i)$ is defined as follows [10, 15]:

$$\tilde{x}_i = \frac{x_i - \mu(\vec{X})}{\sigma(\vec{X})}$$

where $\mu(\vec{X})$ and $\sigma(\vec{X})$ are the mean and standard deviation of the sequence $\vec{X}$, respectively. □

Figure 1 shows an example of normalization transform: Figure 1(a) shows original sequences $\vec{X}$, $\vec{Y}$, and $\vec{Z}$; Figure 1(b) normalized sequences $\nu(\vec{X})$, $\nu(\vec{Y})$, and $\nu(\vec{Z})$. Let us assume that we use the Euclidean distance as the similarity measure. Before transform in Figure 1(a), $\vec{X}$ and $\vec{Z}$ are considered to be similar, but, after transform in Figure 1(b), $\nu(\vec{X})$ and $\nu(\vec{Y})$ are considered to be similar. That is, normalization transform is useful

Table 1: Summary of notation.

Notation	Definition
$\vec{S} = (s_i)$	a data sequence. $\vec{S} = (s_0, \ldots, s_{N-1})$ $(0 \leq i < N)$
$\vec{X} = (x_i)$	a subsequence contained in the data sequence $\vec{S}$. $\vec{X} = (x_0, \ldots, x_{n-1})$ $(0 \leq i < n \leq N)$
$\vec{T} = (t_i)$	the query sequence. $\vec{T} = (t_0, \ldots, t_{n-1})$ $(0 \leq i < n)$
$d(\vec{X}, \vec{T})$	Euclidean distance between two sequences $\vec{X}$ and $\vec{T}$ $(Len(\vec{X}) = Len(\vec{T}))$ $d(\vec{X}, \vec{T}) = \{\sum (x_i - t_i)^2\}^{1/2}$
$\vec{X}[s, f]$	a window consisting of values x_s, ..., x_f in the sequence $\vec{X}$ $(0 \leq s \leq f < n)$
$\vec{Z} = a\vec{X} + b$	a sequence obtained by multiplying each consisting value in $\vec{X}$ by a and then adding b. $\vec{Z} = (z_i) = (ax_i + b)$
ϵ	search range (tolerance)

in finding data sequences that have similar fluctuation patterns.

We now define the problem of subsequence matching that supports normalization transform. Given a query sequence $\vec{T}$ and a search range ϵ, the search is performed using the normalized query sequence $\nu(\vec{T})$. For each data sequence $\vec{S}$ stored in the database, if $\vec{S}$ contains a subsequence $\vec{X}$ that has the same length as that of $\vec{T}$ and that satisfies Eq. (1), $\vec{S}$ and the offset of $\vec{X}$ in $\vec{S}$ are returned.

$$d\left(\nu(\vec{X}), \nu(\vec{T})\right) \leq \epsilon \tag{1}$$

3 Related Work

In this section, we introduce the whole matching algorithm supporting normalization transform [2, 10] and the existing subsequence matching algorithms [9]. We also explain the problems in simply applying these algorithms to normalization transform subsequence matching.

3.1 Subsequence Matching Algorithms

Faloutsos et al. [9] and Agrawal et al. [2] proposed subsequence matching algorithms. The algorithm by Faloutsos et al. is an extension of an earlier work on the whole matching algorithm by Agrawal et al. [1]. The algorithm is performed using a given query sequence $\vec{T}$ of an arbitrary length n and a search range ϵ. For comparing the subsequence $\vec{X}$ of length n in the database with the query sequence $\vec{T}$, $\vec{X}$ and $\vec{T}$ are partitioned into p windows $\vec{\chi}_0$, ..., $\vec{\chi}_{(p-1)}$ and $\vec{\tau}_0$, ..., $\vec{\tau}_{(p-1)}$ of the fixed length w $(n = pw)$ as shown in Figure 2. Faloutsos et al. [9] proved that, if the distance between $\vec{X}$ and $\vec{T}$ is within ϵ, there exists at least one pair of windows $\vec{\chi}_i$ and $\vec{\tau}_i$ $(0 \leq i < p)$ that satisfies Eq. (2):

$$d\left(\vec{\chi}_i, \vec{\tau}_i\right) \leq \frac{\epsilon}{\sqrt{p}} \tag{2}$$

Faloutsos et al. also proposed an algorithm that efficiently searches windows $\vec{\chi}_i$ satisfying Eq. (2) using a multidimensional index.

The algorithm by Faloutsos et al. supports no preprocessing transforms. We cannot simply apply the algorithm to the normalization transform subsequence matching problem because information needed for normalization transform of subsequence $\vec{X}$ is not available from the windows $\vec{\chi}_0$, ..., $\vec{\chi}_{(p-1)}$. The main reason is that each window is stored and processed independently of one another. The algorithm extracts sliding windows $\vec{\chi}_i$ of the fixed length w from the data sequence $\vec{S}$ as shown in Figure 3 and performs indexing and searching. Thus, sliding windows are the unit of indexing and searching. There exist $(n - w + 1)$ subsequences $\vec{X}$ of length n $(\geq w)$, each subsequence having mean and standard deviation values different from one another. However, a window $\vec{\chi}_i$, a part of the subsequence $\vec{X}$, stored in the index does not contain such information for normalization transform, and there does not exist yet a simple method to make the window contain such information.

The algorithm by Agrawal et al. [2] retrieves pairs of data sequences that contain similar normalized subsequences. The normalization transform subsequence matching algorithm proposed in this paper compares the whole normalized subsequences, while the algorithm by Agrawal et al. [2] compares the normalized windows, each window being only a part of the subsequence. Moreover, the proposed algorithm compares the normalized subsequences of arbitrary lengths, while the algorithm by Agrawal et al. [2] compares normalized windows of only a fixed length w. Like the algorithm by Faloutsos et al., the algorithm by Agrawal et al. [2] cannot be simply applied to the normalization transform subsequence matching problem because a sliding window

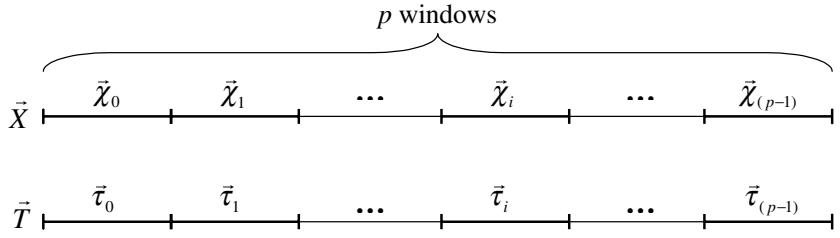


Figure 2: Partitioning the subsequence $\vec{X}$ and the query sequence $\vec{T}$ into p windows.

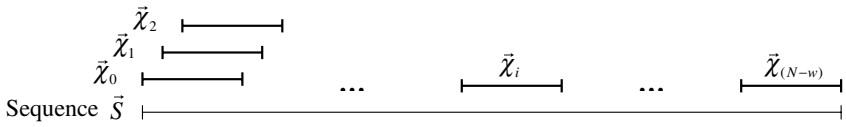


Figure 3: Sliding windows $\vec{\chi}_i$ extracted from a data sequence $\vec{S}$.

$\nu(\vec{\chi}_i)$ stored in the index has no information for normalization transform of a subsequence $\vec{X}$ of length n ($\geq w$) that contains $\vec{\chi}_i$.

3.2 Whole Matching Algorithm Supporting Normalization Transform

Goldin and Kanellakis [10] proposed a whole matching algorithm that supports normalization transform. Goldin and Kanellakis defined that two sequences $\vec{X}$ and $\vec{T}$ of length n are similar if Eq. (3) is satisfied. They also proposed an algorithm that efficiently retrieves data sequences $\vec{X}$ satisfying Eq. (3), given a query sequence $\vec{T}$ and a search range ϵ.

$$\exists a, b(\in R), d\left(\vec{T}, a\vec{X} + b\right) \leq \epsilon \tag{3}$$

Here, R is a set of all real numbers. It is totally inefficient to try to assign arbitrary real numbers a and b to see if a data sequence $\vec{X}$ satisfies the similarity condition of Eq. (3). Therefore, Goldin and Kanellakis [10] presented new search range ϵ' such that normalized sequences $\nu(\vec{X})$ and $\nu(\vec{T})$ satisfy Eq. (4).

$$\exists a, b(\in R), d\left(\vec{T}, a\vec{X} + b\right) \leq \epsilon \Rightarrow d\left(\nu(\vec{T}), \nu(\vec{X})\right) \leq \epsilon' \tag{4}$$

Since the set of data sequences obtained using the consequent of Eq. (4) is a superset of those satisfying the antecedent of Eq. (4), the algorithm causes no false dismissal.

Goldin and Kanellakis discussed only whole matching algorithm, which processes data and query sequences of a fixed length, but presented no extension that applies to subsequence matching. If the algorithm by Goldin and Kanellakis were applied to subsequence matching without any extension, the following problem would occur. In general, for any sequences $\vec{X}$ and $\vec{T}$ of length n and windows $\vec{X}[s, f]$ and $\vec{T}[s, f]$ ($0 \leq s \leq f < n$), Eq. (5) holds [14]:

$$d\left(\vec{X}, \vec{T}\right) \leq \epsilon \Rightarrow d\left(\vec{X}[s, f], \vec{T}[s, f]\right) \leq \epsilon \tag{5}$$

However, for normalized sequences and windows, Eq. (5) does not hold. That is,

$$d\left(\nu(\vec{X}), \nu(\vec{T})\right) \leq \epsilon \not\Rightarrow d\left(\nu(\vec{X}[s, f]), \nu(\vec{T}[s, f])\right) \leq \epsilon \tag{6}$$

Thus, the set of data sequences satisfying the consequent of Eq. (6) is *not* a superset of those satisfying the antecedent. That is, if normalization transform subsequence matching for query sequences of length n were performed using the index generated for query sequences of length $n_0 = f - s + 1$ ($< n$), false dismissal would occur.

4 The Proposed Method

In this section we propose new indexing and search algorithms for subsequence matching supporting normalization transform. Section 4.1 explains the basic ideas for solving the problem. Section 4.2 presents the detailed indexing and search algorithms.

4.1 Basic Ideas

We solve the problem by extending the algorithm by Goldin and Kanellakis [10]. The only method to apply

the algorithm by Goldin and Kanellakis [10] to normalization transform subsequence matching without any extension is to generate an index for each possible query sequence length. It is because, as explained in Section 3.2, an index for the length n_0 ($< n$) cannot be used for a query sequence of length n. However, the method causes serious overhead on storage space and insertion or deletion of data sequences.

To overcome the problem, we propose a new searching method that generates indexes only for a few selected query sequence lengths w and, using them, performs subsequence matching for query sequences of arbitrary lengths n ($\geq w$). We call the index generated for the selected length w *s-index*(w). Given a query sequence of length n, if there exists s-index(n), s-index(n) is used for subsequence matching. Otherwise, we select one of the s-index(w) according to Eq. (7):

$$\omega = \max\{w | w < n\} \tag{7}$$

We need Theorem 1 to search when the query sequence length n is *not* equal to ω. Theorem 1 presents a new search range ϵ' that replaces search range ϵ and guarantees that false dismissal does not occur in the final search result[1]. We omit the proof due to the page limit.

Theorem 1 For the two sequences $\vec{X} = (x_i)$ and $\vec{T} = (t_i)$ $(0 \leq i < n)$ of length n (≥ 1), the following holds $(0 \leq s \leq f < n)$:

$$d\left(\nu(\vec{X}), \nu(\vec{T})\right) \leq \epsilon \Rightarrow d\left(\nu(\vec{X}[s,f]), \nu(\vec{T}[s,f])\right) \leq \epsilon' \tag{8}$$

where ϵ' is defined as in Eq. (9), and ω is the length of window $\vec{T}[s,f]$ $(\omega = f - s + 1)$.

$$\epsilon' = \sqrt{2\omega - 2\sqrt{\omega^2 - \omega \cdot \epsilon^2 \cdot \frac{\sigma^2(\vec{T})}{\sigma^2(\vec{T}[s,f])}}} \tag{9}$$

□

In Eq. (9), the value in the inner square root must be greater than or equal to 0. Therefore, when ϵ' is computed, the condition of Eq. (10) should be first examined. If the condition of Eq. (10) is not satisfied, we should perform the sequential scan since it is not possible to use s-indexes. The value in the outer square root is always greater than or equal to 0.

$$\omega > \epsilon^2 \cdot \frac{\sigma^2(\vec{T})}{\sigma^2(\vec{T}[s,f])} \tag{10}$$

[1] ϵ' can be either greater or less than ϵ.

When the query sequence length n is not equal to ω obtained by Eq. (7), we make a candidate set consisting of all the subsequences that satisfy Eq. (11), the consequent part of Eq. (8), using s-index(ω).

$$d\left(\nu(\vec{X}[s,f]), \nu(\vec{T}[s,f])\right) \leq \epsilon' \tag{11}$$

Since the length of the window $\vec{T}[s,f]$ is ω, there exist $(n-\omega+1)$ windows that can be extracted from the query sequence $\vec{T}$ to perform subsequence matching using Eq. (11). We select one of the windows $\vec{T}[s,f]$ using Eq. (12) $(0 \leq s' \leq f' < n, \omega = f' - s' + 1)$.

$$\vec{T}[s,f] = \left\{ \vec{T}[s',f'] \;\middle|\; \sigma(\vec{T}[s',f']) \text{ is maximum.} \right\} \tag{12}$$

Eq. (12) maximizes the performance of the search using the new search range ϵ'.

Theorem 1 guarantees that the algorithm will not generate any false dismissal. For every subsequence $\nu(\vec{X})$ that satisfies the antecedent part of Eq. (8), the window $\nu(\vec{X}[s,f])$ always satisfies the consequent part. That is, the set of pairs $\left(\nu(\vec{X}[s,f]),\ \nu(\vec{T}[s,f])\right)$ satisfying the consequent part of Eq. (8) is a superset of that of corresponding pairs $\left(\nu(\vec{X}),\ \nu(\vec{T})\right)$ satisfying the antecedent part. Thus, the search performed using the consequent part does not cause false dismissal.

4.2 Indexing and Searching Algorithms

In this section we explain the algorithms for generating s-index(w) and for searching using it. The s-index(w) is generated in the same way as in the algorithm by Goldin and Kanellakis [10], except that normalized sliding windows are stored in the index. As shown in Figure 3, from a data sequence $\vec{S}$ of length N, $(N-w+1)$ sliding windows $\vec{\chi}_i = \vec{X}[i, i+w-1]$ $(0 \leq i \leq N-w, 0 \leq j < w)$ of length w are extracted. For each normalized sliding window $\nu(\vec{\chi}_i)$, a record $\left(\phi_{i0},\ \cdots,\ \phi_{i(f-1)}\right)$ is stored in an f-dimensional index structure. Here, the values ϕ_{i0}, $\cdots$, $\phi_{i(f-1)}$ are the first f ($< w$) non-zero coefficients obtained through the DFT transform on the component values $\tilde{\chi}_{i0}$, $\cdots$, $\tilde{\chi}_{i(w-1)}$ of a normalized sliding window $\nu(\vec{\chi}_i)$ [1, 9, 10].

The search algorithm using s-index(w) is as follows. If there exists w of s-index(w) that is equal to the query sequence length n, we perform the same search algorithm as the one by Goldin and Kanellakis [10]. In this section we are devoted to the case when there exists no w that is equal to n. The algorithm *Normalized-Search*($\vec{T}$, ϵ) in Figure 4 retrieves subsequences $\vec{X}$ that satisfy Eq. (1) given query sequence $\vec{T}$ and search range ϵ. The algorithm can use any kind of multidimensional index structure.

Procedure *NormalizedSearch*(Sequence $\vec{T}$, Range ϵ)

// Passed parameters
Sequence $\vec{T}$; // query sequence
Range ϵ; // search range

(1) Find ω and $\vec{T}[s, f]$;
(2) **if** Eq. (10) is satisfied **then**
(3) Compute ϵ';
(4) Perform range search using $\nu(\vec{T}[s, f])$ and ϵ';
(5) Make a candidate set C
consisting of the windows returned;
(6) **else**
(7) Make a candidate set C
consisting of all the windows in database;
(8) **endif**
(9) **for** each window $\nu(\vec{X}[s, f])$ in C **do**
(10) Read the subsequence $\vec{X}$ from disk;
(11) **if** $d\left(\nu(\vec{X}), \nu(\vec{T})\right) \leq \epsilon$ **then** return $\vec{X}$;
(12) **end for**

Figure 4: Normalization Transform Subsequence Matching Algorithm

5 Performance Evaluation

In this section we present the experimental results of the proposed algorithm. The result shows that the performance for the selectively-indexed case is comparable to that for the fully-indexed case and that the search performance gets better as the number of s-index(w) increases. The result also shows that the proposed algorithm outperforms the sequential scan algorithm. The search algorithm for the fully-indexed case is identical to the algorithm by Goldin and Kanellakis [10]. We present the environment for experiments in Section 5.1 and the experimental results and analyses in Section 5.2.

5.1 Environments for Experiments

The time-series database used in the experiments consists of 620 data sequences of Korean stock items of length 1024 dated from November 1, 1994 to May 30, 1998. We have generated, for the fully-indexed case, the indexes for the query sequence lengths $n = 256 + 32i$ ($i = 0, \ldots, 8$) and $n = n' \pm 1$ ($n' = 256$, 320, 384, 448, and 512). For the selectively-indexed case, we have used only those of the lengths n'. We have used the indexes for $n = n' \pm 1$ to observe how the search performance changes as the query sequence length changes from $n = n' - 1$ to $n = n' + 1$. We have used the DFT transform to reduce the dimension of s-index(w), and set the dimensionality f of the index to 6, which shows the best search performance. To generate the query sequences $\vec{T}$, we have randomly chosen 128 out of 620 data sequences, and from them, randomly extracted subsequences $\vec{Q} = (q_i)$ ($0 \leq i < 256$) of length 256 as in the reference [9]. We then have generated the query sequences $\vec{T} = (t_i)$ ($0 \leq i < 256$) by perturbing each q_i as in Eq. (13) [1].

$$t_i = q_i + z_i, \ z_i \in (-50, 50) \tag{13}$$

Here, z_i is an arbitrary value in the range $(-50, 50)$, and 50 represents 5 % of the average of $|q_{i+1} - q_i|$ ($0 \leq i < 255$) for all $\vec{Q}$. We set search ranges ϵ so that the final search result using ϵ should satisfy the *selectivity* defined in Eq. (14). We use the selectivity values 0.00001, 0.0001, 0.001, and 0.01.

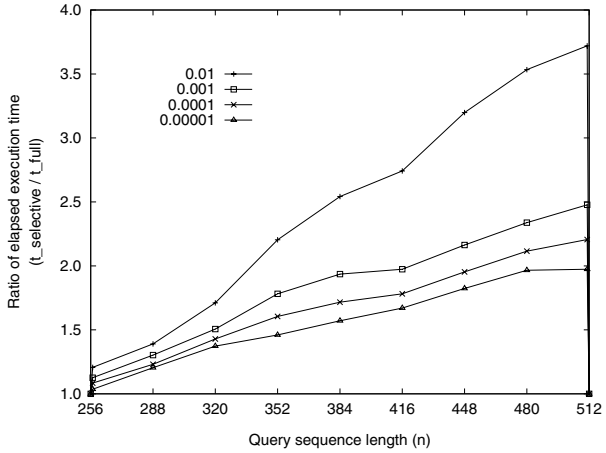
$$\text{Selectivity} = \frac{\#\text{ subsequences in the final result}}{\#\text{ all the possible subsequences in the database}} \tag{14}$$

We have used the R*-tree [4] as the multidimensional index structure to store sliding windows. The hardware platform for the experiment is a PC equipped with an Intel Celeron 400 MHz CPU, 128 MB RAM, and a 2.0 GB Hard Disk. The software platform is Microsoft Korean Windows NT Workstation 4.0 Operating System (OS). To avoid buffering effects of OS and to guarantee actual disk I/Os, we have used OS functions for raw disk access of data and index files [13].

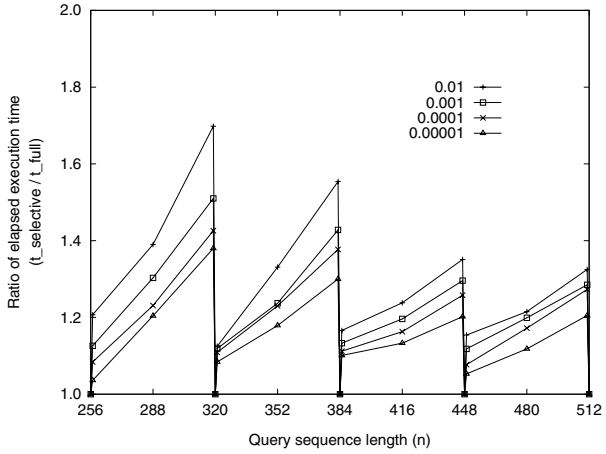
5.2 Experimental Results and Analyses

The first experiment compares the elapsed times of the algorithm execution for selectively-indexed and fully-indexed cases. The purpose of the experiment is to show that the performance of the proposed algorithm does not degrade much even in selectively-indexed cases. Figure 5 shows the result of the experiment. Figure 5(a) shows the result using two s-indexes (w = 256, 512); Figure 5(b) using five s-indexes (w = 256, 320, 384, 448, 512). The vertical axis represents the ratio of the execution time for the selectively-indexed case, $t_{selective}$, divided by that for the fully-indexed case, t_{full}. Each value has been averaged for 128 queries. The result shows that a better search performance is obtained with more s-indexes and that there is no significant degradation of the ratio in selectively-indexed case when using five s-indexes (The maximum ratio is less than 1.70.).

The second experiment compares the elapsed times for the proposed algorithm and the sequential scan algorithm. Figure 6(a) shows the result using two s-indexes (w = 256, 512); Figure 6(b) using five s-indexes (w = 256, 320, 384, 448, 512). The vertical axis represents the

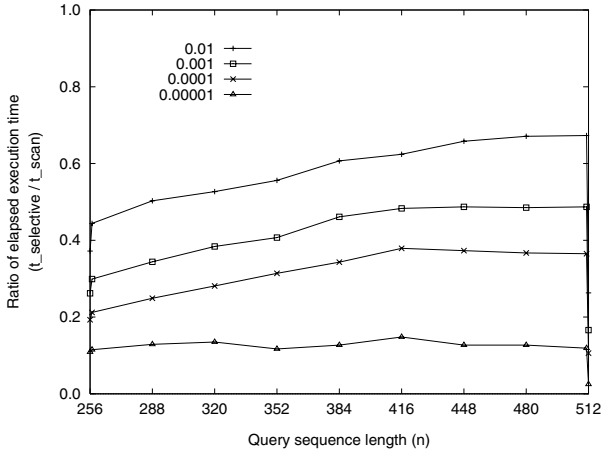


(a) Using two s-indexes.

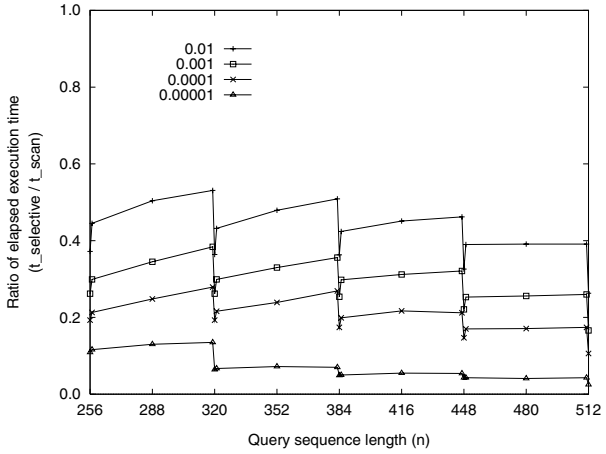


(b) Using five s-indexes.

Figure 5: The ratio of the execution time for the selectively-indexed case divided by that of the fully-indexed case ($t_{selective}/t_{full}$).



(a) Using two s-indexes.



(b) Using five s-indexes.

Figure 6: The ratio of the execution time of the proposed algorithm divided by that of the sequential scan algorithm ($t_{selective}/t_{scan}$).

ratio of the execution time of the proposed algorithm using s-index(w), $t_{selective}$, divided by that of the sequential scan algorithm, t_{scan}. Each value has been averaged for 128 queries. When using five s-indexes, the average ratio of search performance by the proposed algorithm compared with that of the sequential scan algorithm is 2.40 $(\frac{1}{0.42})$, 3.49 $(\frac{1}{0.29})$, 4.97 $(\frac{1}{0.20})$, and 14.6 $(\frac{1}{0.07})$ times for selectivities 0.01, 0.001, 0.0001, and 0.00001, respectively.

In general, the queries with smaller selectivities are much more frequent than those with larger ones in most of database applications. As shown in Figures 5 and 6, the search performance of the proposed algorithm gets better as the selectivity decreases. This makes the proposed algorithm more useful in practical database application environments, where small selectivities prevail.

6 Conclusions

In this paper, we have proposed an efficient subsequence matching algorithm that supports normalization transform. We have discussed that the existing subsequence matching algorithms [2, 9] cannot simply be applied to the normalization transform subsequence matching problem because the algorithms do not have information for normalization transform of the arbitrary length subsequences. We have also discussed that, to use the existing whole matching algorithm supporting normalization transform [10], we need to generate an index for each possible query sequence length and that this would cause serious overhead on storage space and insertion or deletion of data sequences.

The proposed algorithm solves the problem by us-

ing index interpolation. Given a search range ϵ, we have presented the distance boundary ϵ' between two windows each extracted from the query sequence and a data sequence, respectively, and have shown that ϵ' is a function of only the query sequence instead of the data sequence stored in the index. The proposed algorithm generates s-indexes only for a few pre-selected query sequence lengths w with some proper intervals, and using them, performs subsequence matching for arbitrary lengths n of query sequences. We have proved that the proposed algorithm is correct in that it does not cause false dismissal.

We have evaluated the performance of the proposed algorithm by experiments. The results show that the performance of the proposed algorithm for selectively-indexed cases is comparable to that for fully-indexed cases and that better search performance is obtained when more s-indexes are used. When s-indexes for five query sequence lengths chosen among the lengths $256 \sim 512$ are used, the proposed algorithm outperforms the sequential scan algorithm on the average by up to 2.4 times when the selectivity is 10^{-2} and by up to 14.6 times when the selectivity is 10^{-5}. The performance is enhanced by using more s-indexes. The performance also gets better when the selectivity is smaller. This makes the algorithm more useful in practical database applications.

References

[1] Agrawal, R. et al., "Efficient Similarity Search in Sequence Databases," In *Proc. Int'l Conf. on Foundations of Data Organization and Algorithms*, pp. 69-84, Chicago, Illinois, Oct. 1993.

[2] Agrawal, R. et al., "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*, pp. 490-501, Zurich, Switzerland, Sept. 1995.

[3] Agrawal, R. et al., "Querying Shapes of Histories," In *Proc. Int'l Conf. on Very Large Data Bases*, pp. 502-514, Zurich, Switzerland, Sept. 1995.

[4] Beckmann, N. et al., "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, pp. 322-331, Atlantic City, NJ, June 1990.

[5] Berchtold, S. et al., "The X-tree: An Index Structure for High-Dimensional Data," In *Proc. Int'l Conf. on Very Large Data Bases*, pp. 28-39, Mumbai, India, Sept. 1996.

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

[7] Chan, K.-P. and Fu, W.-C., "Efficient Time Series Matching by Wavelets," In *Proc. Int'l Conf. on Data Engineering*, IEEE, pp. 126-133, Sydney, Australia, Mar. 1999.

[8] Chu, K. K. W. and Wong, M. H., "Fast Time-Series Searching with Scaling and Shifting," In *Proc. ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp. 237-248, Philadelphia, Pennsylvania, May 1999.

[9] Faloutsos, C. et al., "Fast Subsequence Matching in Time-Series Databases," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, pp. 419-429, Minneapolis, Minnesota, June 1994.

[10] Goldin, D. Q. and Kanellakis, P. C., "On Similarity Queries for Time-Series Data: Constraint Specification and Implementation," In *Proc. Int'l Conf. on Principles and Practices of Constraint Programming*, pp. 137-153, Cassis, France, Sept. 1995.

[11] Gonzalez, R. C. and Woods, R. E., *Digital Image Processing*, Addison-Wesley, 1993.

[12] Guttman, A., "R-trees: A Dynamic Index Structure for Spatial Searching," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, pp. 47-57, Boston, Massachusetts, June 1984.

[13] Hart, J. M., *Win32 System Programming*, Addison-Wesley Developers Press, 1997.

[14] Press, W. H. et al., *Numerical Recipes in C – The Art of Scientific Computing*, 2nd Ed., Cambridge University Press, 1992.

[15] Rafiei, D. and Mendelzon, A., "Similarity-Based Queries for Time Series Data," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, pp. 13-25, Tucson, Arizona, June 1997.

[16] Sellis, T. et al., "The R^+-tree: A Dynamic Index for Multidimensional Objects," In *Proc. Int'l Conf. on Very Large Data Bases*, pp. 507-518, Brighton, England, Sept. 1987.

[17] Weber, R. et al., "A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces," In *Proc. Int'l Conf. on Very Large Data Bases*, pp. 194-205, New York, New York, Aug. 1998.

[18] Yi, B.-K. et al., "Efficient Retrieval of Similar Time Sequences Under Time Warping," In *Proc. Int'l Conf. on Data Engineering*, IEEE, pp. 201-208, Orlando, Florida, Feb. 1998.