IEICE TRANS. INF. & SYST., VOL.E84-D, NO.1 JANUARY 2001

PAPER

Index Interpolation: A Subsequence Matching Algorithm Supporting Moving Average Transform of Arbitrary Order in Time-Series Databases*

Woong-Kee LOH[†], Sang-Wook KIM[††], *and* Kyu-Young WHANG[†], *Nonmembers*

SUMMARY In this paper we propose a subsequence matching algorithm that supports moving average transform of arbitrary order in time-series databases. Moving average transform reduces the effect of noise and has been used in many areas such as econometrics since it is useful in finding the overall trends. The proposed algorithm extends the existing subsequence matching algorithm proposed by Faloutsos et al. (SUB94 in short). If we applied the algorithm without any extension, we would have to generate an index for each moving average order and would have serious storage and CPU time overhead. In this paper we tackle the problem using the notion of index interpolation. *Index interpolation* is defined as a searching method that uses one or more indexes generated for a few selected cases and performs searching for all the cases satisfying some criteria. The proposed algorithm, which is based on index interpolation, can use only one index for a pre-selected moving average order k and performs subsequence matching for arbitrary order m ($\leq k$). We prove that the proposed algorithm causes no false dismissal. The proposed algorithm can also use more than one index to improve search performance. The algorithm works better with smaller selectivities. For selectivities less than 10^{-2}, the degradation of search performance compared with the fully-indexed case—which is equivalent to SUB94—is no more than 33.0% when one index is used, and 17.2% when two indexes are used. Since the queries with smaller selectivities are much more frequent in general database applications, the proposed algorithm is suitable for practical situations.

key words: *index interpolation, subsequence matching, moving average transform, time-series databases*

1. Introduction

Time-series data are the sequences of real numbers sampled at a fixed time interval. 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 [8], [17]. 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]–[3], [8], [17].

Similar sequence matching algorithms are classified into whole matching or subsequence matching [8]. 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 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], [17], [20], although some algorithms use different similarity measures [2]. The algorithms use multidimensional index structures such as the R-tree [11], R$^+$-tree [18], and R*-tree [4] to efficiently store and retrieve the n-dimensional points. Since the search performance degrades exponentially as the dimensionality of the index structures increases [5], [19], most of the existing algorithms reduce the dimensionality by mapping the n-dimensional points into the f-dimensional ones ($f < n$). The Discrete Fourier Transform (DFT) [16], Discrete Cosine Transform (DCT) [16], and Haar Wavelet Transform [10] are used as the mapping functions for dimensionality reduction [1], [7]–[9], [17].

Existing algorithms can also be classified according to the types of pre-processing transform, which is performed before any comparison between data and query sequences. The algorithms proposed by Agrawal et al. [1] and Faloutsos et al. [8] perform no pre-processing transform and the algorithms proposed by Agrawal et al. [2], Goldin and Kanellakis [9], Rafiei and Mendelzon [17], and Yi et al. [20] perform scaling, shifting, nor-

Manuscript received November 24, 1999.
Manuscript revised April 21, 2000.

[†]The authors are with the Electronic Engineering and Computer Science Department, Division of Computer Science and Advanced Information Technology Research Center (AITrc), Korea Advanced Institute of Science and Technology (KAIST), Korea.

[††]The author is with the Division of Computer, Information, and Communications, Kangwon National University, Korea.

*This work has been partially supported by Ministry of Science and Technology through the project "Development of a Web-DBMS Integration Technology for a Web-based OODBMS Supporting Information Retrieval" and by Korea Science and Engineering Foundation (KOSEF) through Advanced Information Technology Research Center (AITrc).

malization, moving average, and time warping transforms. The purpose of the pre-processing transform is to give flexibility in the definition of sequence similarity to satisfy specific application needs.

In this paper we propose a subsequence matching algorithm that efficiently supports moving average transform [6], [12], [17] of arbitrary order. The moving average transform converts a given data sequence into a new sequence consisting of the averages of k consecutive values in the data sequence, where k is called the *moving average order* or simply the *order*. The moving average transform is very useful for finding the trend of the time-series data by reducing the effect of noise inside, and has been used in various applications [6]. Since the users want to control the degree of the noise reduction depending on the characteristics of data sequences to be analyzed, the proper moving average order varies depending on the applications [12]. For example, the moving average orders of 6, 25, 75, and 150 are frequently used to find the trend of stock prices, where the smaller ones are used for shorter term analysis and the larger ones for longer term analysis. Thus, efficient support of arbitrary orders is necessary.

The simplest method for supporting moving average transform of arbitrary order is just to apply the existing subsequence matching algorithm proposed by Faloutsos et al. [8] without any extension. We simply call this algorithm *SUB94* in this paper. Because this method would require one index per moving average order, however, it causes serious overhead in storage space and insertion or deletion of data sequences. We tackle the problem using the notion of *index interpolation* defined as follows:

Definition 1: *Index interpolation* is a searching method that uses one or more indexes for a few selected cases and performs searching for all the cases satisfying some criteria. □

The proposed algorithm requires only one index for a selected moving average order k and performs subsequence matching for arbitrary order m ($\leq k$). It can also use more than one index to improve search performance; the more indexes, the better the search performance. We prove that the proposed algorithm does not cause false dismissal. With experiments, we show that the search performance with only one index does not degrade significantly compared with that with indexes for every moving average order, and that the search performance improves as we use more indexes. We call the case with the indexes for the selected moving average orders as *selectively-indexed case* and the one for all the orders as *fully-indexed case*.

This paper is organized as follows: First, we formally define the problem of subsequence matching that supports moving average transform in Sect. 2. In Sect. 3, we briefly introduce the existing subsequence matching algorithm [8] and the problems in applying it to support moving average transform of arbitrary order. In Sect. 4, we present the proposed algorithm. In Sect. 5, we evaluate the performance of the proposed algorithm by extensive experiments. Finally, we summarize and conclude the paper in Sect. 6.

2. Problem Definition

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

Definition 2: Given a sequence $\vec{X} = (x_i)$ $(0 \leq i < n)$ and a moving average order k $(1 \leq k \leq n)$, the *k-moving average transformed sequence* $\vec{X}_{(k)} = (x_{(k)j})$ $(0 \leq j < n - k + 1)$ is defined as follows [6], [12]:

$$x_{(k)j} = \frac{1}{k}(\overbrace{x_j + \cdots + x_{j+k-1}}^{k}) = \frac{1}{k}\sum_{i=j}^{j+k-1} x_i$$

where n is the length of data sequence $\vec{X}$. □

Figure 1 shows an example of original and moving average transformed sequences. The length of the sequence $\vec{X}$ before transformation is 32; and the lengths of the 4- and 8-moving average transformed sequences, $\vec{X}_{(4)}$ and $\vec{X}_{(8)}$, are 29 (= 32−4+1) and 25 (= 32−8+1),

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 $\vec{S}$. $\vec{X} = (x_0, \ldots, x_{n-1})$ $(0 \leq i < n \leq N)$
$\vec{X}_{(k)}$	the k-moving average transformed result of the sequence $\vec{X}$
$\vec{T} = (t_i)$	the query sequence. $\vec{T} = (t_0, \ldots, t_{n-1})$ $(0 \leq i < n)$
$\vec{T}_{(k)}$	the k-moving average transformed result of the query sequence $\vec{T}$
$d(\vec{X}, \vec{T})$	the Euclidean distance between two sequences $\vec{X}$ and $\vec{T}$ $(Len(\vec{X}) = Len(\vec{T}))$. $d(\vec{X}, \vec{T}) = \{\sum(x_i - t_i)^2\}^{1/2}$
$\vec{Z} = \vec{X} + d_s$	a sequence obtained by shifting the sequence $\vec{X}$ by d_s. $\vec{Z} = (z_i) = (x_i + d_s)$
ϵ	search range (tolerance)
w	the length of (sliding) windows
w_k	the length of the k-moving average transformed windows
d_s	the shifting distance

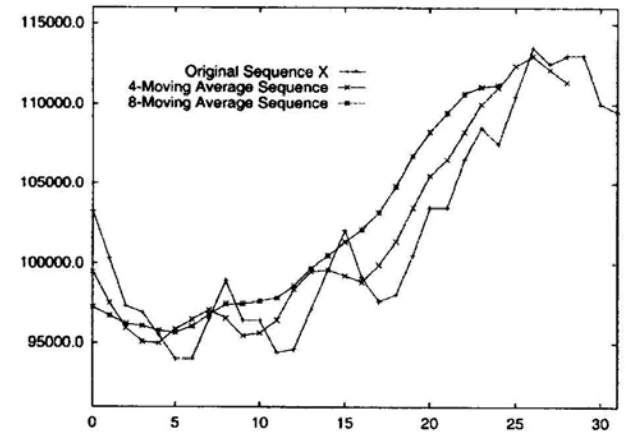


Fig. 1 An example of original and moving average transformed sequences.

respectively. We can see that the effect of noise decreases as the moving average order increases.

The problem of subsequence matching that supports moving average transform of arbitrary order is defined as follows: Given a query sequence $\vec{T}$ and a moving average order k, the search is performed using the k-moving average transformed sequence $\vec{T}_{(k)}$ of $\vec{T}$. For the k-moving average transformed sequence $\vec{S}_{(k)}$ of each data sequence $\vec{S}$ stored in a database, if $\vec{S}_{(k)}$ contains a subsequence $\vec{X}_{(k)}$ that is similar to and has the same length as $\vec{T}_{(k)}$, $\vec{S}$ and the offset of $\vec{X}_{(k)}$ in $\vec{S}_{(k)}$ are returned.

3. Related Work

In Sect. 3.1, we briefly explain SUB94 [8] on which the proposed algorithm is based. In Sect. 3.2, we introduce a whole matching algorithm supporting moving average transform [17], which we call *MOV97* in this paper, and describe its problems.

3.1 The SUB94 Algorithm

We first explain the properties of DFT to be used in the SUB94 algorithm [8]. First, for any two sequences $\vec{X}$ and $\vec{T}$, the Euclidean distance between $\vec{X}$ and $\vec{T}$ is identical to that between their DFT transformed results $F(\vec{X})$ and $F(\vec{T})$, i.e., it holds that $d\left(\vec{X},\vec{T}\right) = d\left(F(\vec{X}),F(\vec{T})\right)$ (Parseval's Theorem [1], [7], [8], [17]). Second, most of the energy[†]of $F(\vec{X})$, the DFT transformed result of $\vec{X}$, is contained in a few of the first coefficients of $F(\vec{X})$, i.e., $\|F(\vec{X})\|^2 \approx \|\hat{F}(\vec{X})\|^2$, where $\hat{F}(\vec{X})$ is a sequence consisting of a few of the first coefficients of $F(\vec{X})$.

The multidimensional index structures are used to efficiently store and find points whose dimensionality is greater than one. Since the search performance using the multidimensional index structures degrades exponentially as the dimensionality increases [5], [19], the SUB94 algorithm stores in the index the coefficients of $\hat{F}(\vec{X})$ instead of those of $\vec{X}$. Since it holds that $d\left(\vec{X},\vec{T}\right) \geq d\left(\hat{F}(\vec{X}),\hat{F}(\vec{T})\right)$ [8], every sequence pair $\vec{X}$ and $\vec{T}$ that satisfies $d\left(\vec{X},\vec{T}\right) \leq \epsilon$ also satisfies $d\left(\hat{F}(\vec{X}),\hat{F}(\vec{T})\right) \leq \epsilon$. Thus, the use of the index storing $\hat{F}(\vec{X})$ instead of $\vec{X}$ does not cause false dismissal.

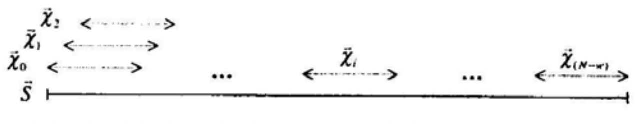


Fig. 2 Sliding windows extracted from data sequence.

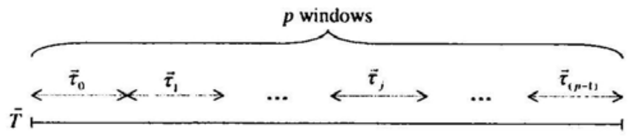


Fig. 3 Partitioning query sequence into p windows.

The SUB94 algorithm performs subsequence matching in two phases. In the indexing phase, from each data sequence of length N, sliding windows $\vec{\chi}_i$ $(0 \leq i \leq N-w)$ of length w $(\leq N)$ are extracted as shown in Fig. 2. Next, for each window $\vec{\chi}_i$, we store $(\phi_{i0},\ldots,\phi_{i(f-1)})$ in the f-dimensional index structure, where the values $\phi_{i0},\ \ldots,\ \phi_{i(f-1)}$ are chosen from the DFT result of the values $\chi_{i0},\ \ldots,\ \chi_{i(w-1)}$ constituting $\vec{\chi}_i$ so that the chosen values contain most of the energy of $\vec{\chi}_i$. In the search phase, given a search range ϵ, the query sequence $\vec{T}$ of length n $(\leq N)$ is partitioned into p windows $\vec{\tau}_j$ $(0 \leq j < p)$ of length w as shown in Fig. 3. The search algorithm then retrieves every window $\vec{\chi}_i$ satisfying Eq. (1) below for every window $\vec{\tau}_j$ in the query sequence using the index generated in the indexing phase. When accessing the index, the algorithm uses the f $(< w)$ coefficients obtained from each window $\vec{\tau}_j$ through DFT in the same way as in the indexing phase.

$$d(\vec{\chi}_i,\vec{\tau}_j) \leq \frac{\epsilon}{\sqrt{p}} \tag{1}$$

The candidate set consists of the subsequences that contain the resulting windows. The subsequences in the candidate set are read from disk; only those whose Euclidean distance from the query sequence is within the ϵ range are returned as the final result. Faloutsos et al. [8] proved that SUB94 does not cause false dismissal, i.e., it does not miss a part of the final result.

3.2 The MOV97 Algorithm

The MOV97 algorithm by Rafiei and Mendelzon [17]

[†]The *energy* $E(\vec{X})$ of a sequence $\vec{X}$ is defined as $E(\vec{X}) \equiv \|\vec{X}\|^2 \equiv \Sigma x_i^2$.

is a similar sequence matching algorithm that employs the convolution definition [16] to support moving average transform of arbitrary order using only one index. MOV97 uses the definition of moving average transform that is different from the traditional one [6], [12] presented in Sect. 2. Based on Definition 2, when we perform moving average transform of order k on a sequence of length n ($\geq k$), we cannot compute k-moving average values for the last $(k-1)$ values of the sequence. But MOV97 performs the transform using the first $(k-1)$ values in a circular manner. This new definition enables MOV97 to use convolution to perform moving average transform.

MOV97 is a whole matching algorithm where the lengths of data sequences and the query sequence are all identical. Rafiei and Mendelzon [17] do not mention any extension of their algorithm for subsequence matching. When we try to combine MOV97 with SUB94 [8] to support subsequence matching, we encounter a problem due to the non-traditional definition of moving average transform used in MOV97. That is, it does not even make sense to apply MOV97 to subsequence matching. It is because, when performing k-moving average transform on a sequence $\vec{X}$ using the definition of MOV97, the result obtained by transforming the whole sequence $\vec{X}$ is quite different from that obtained by separately transforming each window $\vec{\chi}_i$ extracted from the sequence $\vec{X}$. When the whole sequence $\vec{X}$ of length n ($\geq k$) is k-moving average transformed, the front $(k-1)$ values of the sequence $\vec{X}$ is used for computing the last $(k-1)$ moving average values. On the other hand, when each window $\vec{\chi}_i$ is k-moving average transformed separately, the front $(k-1)$ values of the each window $\vec{\chi}_i$ rather than those of the whole sequence $\vec{X}$ are used. In addition, the transformed results get different according to the window length w.

4. The Proposed Method

In this section we propose new indexing and search algorithms for subsequence matching supporting moving average transform of arbitrary order. In Sect. 4.1 we explain the basic ideas for solving the problem. In Sect. 4.2 we present the detailed indexing and search algorithms.

4.1 Basic Ideas

In this paper we solve the problem by extending SUB94. For applying SUB94 without any extension to the problem, we must generate an index for each moving average order for the following reason. SUB94, not taking preprocessing transforms into account, performs indexing and searching in the unit of sliding windows $\vec{\chi}_i$ of fixed length w extracted from a data sequence $\vec{S}$. According to Definition 2, when performing k-moving average transform on a sliding window $\vec{\chi}_i$ to get a transformed window $\vec{\chi}_{(k)i}$ of length w, we need $(k-1)$ values following the sliding window $\vec{\chi}_i$ in the original data sequence $\vec{S}$. For example, in Fig. 1, to get the 4-moving average transformed sequence $\vec{X}_{(4)}$ of the sequence $\vec{X}$, we need three more values that follow $\vec{X}$. However, each sliding window $\vec{\chi}_i$ is treated as an independent unit and has no information on other sliding windows even though they are extracted from the same data sequence. That is, the sliding window $\vec{\chi}_i$ cannot be k-moving average transformed to be compared with the query window $\vec{\tau}_{(k)i}$ of length w. Thus, the only way to apply SUB94 is to perform indexing and searching on the sliding windows $\vec{\chi}_{(k)i}$ extracted from the moving average transformed sequence $\vec{S}_{(k)}$. To support moving average transform of arbitrary order, however, this method must generate one index for each moving average order. This results in serious overhead in storage space and insertion or deletion of data sequences.

We approach the problem by dividing it into two cases: the plain search case and the index interpolation search case. In the plain search case, we solve the problem of subsequence matching only for the moving average order for which an index is generated; in the index interpolation search case, we solve it for an arbitrary moving average order. In the index interpolation search case, we generate only one index for a moving average order k and perform searching for an arbitrary moving average order m ($\leq k$). We also explain the case of using the indexes for more than one moving average order for improving search performance.

The plain search case is a simple application of SUB94. In the indexing phase, we first generate the k-moving average transformed sequence $\vec{S}_{(k)}$ of the data sequence $\vec{S}$ for a given order k. The length of $\vec{S}_{(k)}$ is $N-k+1$, where N is the length of $\vec{S}$. Next, we extract sliding windows $\vec{\chi}_{(k)i}$ of the predetermined length w_k out of $\vec{S}_{(k)}$ as shown in Fig. 2, and store f ($< w_k$) coefficients obtained through DFT for each $\vec{\chi}_{(k)i}$ in an f-dimensional index structure. In the search phase, given the query sequence $\vec{T}$ and a search range ϵ, we first generate the k-moving average transformed sequence $\vec{T}_{(k)}$. The length of $\vec{T}_{(k)}$ is $n-k+1$, where n ($\geq k$) is the length of $\vec{T}$. Next, we partition $\vec{T}_{(k)}$ into p disjoint windows $\vec{\tau}_{(k)j} (0 \leq j < p)$ of length w_k as shown in Fig. 3. We use each window $\vec{\tau}_{(k)j}$ to retrieve the candidate set consisting of subsequences that contain windows $\vec{\chi}_{(k)i}$ satisfying the following equation:

$$d\left(\vec{\chi}_{(k)i}, \vec{\tau}_{(k)j}\right) \leq \frac{\epsilon}{\sqrt{p}}$$

For each subsequence in the candidate set, we read the subsequence from the disk and check the distance from the query sequence is within ϵ to include it in the final result.

We now explain the index interpolation search case. In the indexing phase, only one index is generated for a pre-selected moving average order k, where k is estimated to be the maximum moving average order expected in the queries. Additional information is stored in the index to handle the queries with moving average orders $m \le k$ for which no index is generated. We call the index generated for the moving average order k the *k-index*. We explain how to generate the k-index in detail in Sect. 4.2.

In the search phase, given a query sequence $\vec{T}$, a moving average order m, and a search range ϵ, the searching should be performed using the m-moving average transformed sequence $\vec{T}_{(m)}$. That is, as in the plain search case, we construct the candidate set consisting of subsequences that contain windows $\vec{\chi}_{(m)i}$ satisfying Eq. (2):

$$d\left(\vec{\chi}_{(m)i}, \vec{\tau}_{(m)j}\right) \le \frac{\epsilon}{\sqrt{p}} \tag{2}$$

When the given moving average order m is equal to the order k of the k-index, the searching is performed using the k-index in the same way as in the plain search case. For searching when m is less than k, we first introduce Theorem 1 and Corollary 1.

Theorem 1: For sequences $\vec{X} = (x_i)$ and $\vec{T} = (t_i)$ $(0 \le i < n)$ of length n,

$$d\left(\vec{X}_{(m)}, \vec{T}_{(m)}\right) \ge d\left(\vec{X}_{(k)}, \vec{T}_{(k)}\right)$$

if $1 \le m \le k < n-1$ $(n \ge 3)$ and the values in either sequence are all greater than the corresponding values in the other one, i.e., the following condition is satisfied:

$$\forall i, x_i \ge t_i \ \vee \ \forall i, x_i \le t_i \ (0 \le i < n)$$

Proof: Refer to [14]. □

Corollary 1: For windows $\vec{\chi}_i = (\chi_{il})$ and $\vec{\tau}_j = (\tau_{jl})$ $(0 \le l < w)$ of length w, the following relationship holds:

$$d\left(\vec{\chi}_{(m)i}, \vec{\tau}_{(m)j}\right) \le \frac{\epsilon}{\sqrt{p}} \Rightarrow d\left(\vec{\chi}_{(k)i}, \vec{\tau}_{(k)j}\right) \le \frac{\epsilon}{\sqrt{p}} \tag{3}$$

if $1 \le m \le k < w-1$ $(w \ge 3)$ and the values in either window are all greater than the corresponding values in the other one, i.e., the following condition is satisfied (called the *condition of non-overlapping windows* in this paper):

$$\forall l, \chi_{il} \ge \tau_{jl} \ \vee \ \forall l, \chi_{il} \le \tau_{jl} \ (0 \le l < n) \tag{4}$$

□

If the order m given at query time is *not* equal to the order k of the k-index, we retrieve the candidate set satisfying Eq. (5) instead of Eq. (2) using Corollary 1.

$$d\left(\vec{\chi}_{(k)i}, \vec{\tau}_{(k)j}\right) \le \frac{\epsilon}{\sqrt{p}} \tag{5}$$

Here, the condition of non-overlapping windows in Eq. (4) must be satisfied. The case when the condition is not satisfied is dealt with later in this section. Corollary 1 has other conditions: $w \ge 3$ and $1 \le k < w-1$. However, the application scope of the proposed method is not limited by the condition, since we can always set k and w to satisfy the condition.

In Definition 3, we define the notion of *matching windows* to perform the subsequence matching for arbitrary order:

Definition 3: The window $\vec{\chi}_{(k)i}$ is called the *matching window* of $\vec{\chi}_{(m)i}$, where $\vec{\chi}_{(k)i}$ and $\vec{\chi}_{(m)i}$ are k- and m-moving average transformed windows of $\vec{\chi}_i$, respectively. □

Matching windows are used when the order m given at query time is not equal to the order k of the k-index. That is, the matching window $\vec{\tau}_{(k)j}$ instead of the window $\vec{\tau}_{(m)j}$ is used for searching, and the matching window $\vec{\chi}_{(k)i}$ instead of the window $\vec{\chi}_{(m)i}$ is returned as the search result.

Given a window $\vec{\chi}_{(m)i}$, we can easily find the matching window $\vec{\chi}_{(k)i}$, and *vice versa*, using the following two properties. (1) Offset Property: The offset of the window $\vec{\chi}_i$ in the subsequence $\vec{X}$ and the offsets of window $\vec{\chi}_{(m)i}$ and its matching window $\vec{\chi}_{(k)i}$ in the m- and k-moving average transformed sequences $\vec{X}_{(m)}$ and $\vec{X}_{(k)}$ are all the same. (2) Length Property: According to Definition 2, the difference between the lengths of window $\vec{\chi}_{(m)i}$ and its matching window $\vec{\chi}_{(k)i}$ is easily computed using Eq. (6) below:

$$w_m - w_k = k - m \tag{6}$$

where w_m and w_k are the lengths of $\vec{\chi}_{(m)i}$ and $\vec{\chi}_{(k)i}$, respectively. Figure 4 shows the relationship of offsets and lengths of the window $\vec{\chi}_i$, m-moving average transformed window $\vec{\chi}_{(m)i}$, and its matching window $\vec{\chi}_{(k)i}$. The length w of $\vec{\chi}_i$ is $w = w_k + k - 1$.

We can show by Corollary 1 that the search using Eq. (5) and the k-index does not cause false dismissal. For the windows $\vec{\chi}_{(m)i}$ that satisfy the antecedent in Eq. (3), the matching windows $\vec{\chi}_{(k)i}$ satisfy the consequent. That is, the set of pairs $\left(\vec{\chi}_{(k)i}, \ \vec{\tau}_{(k)j}\right)$ that satisfy the consequent is a superset of the set of pairs $\left(\vec{\chi}_{(m)i}, \ \vec{\tau}_{(m)j}\right)$ that satisfy the antecedent. Thus, the search using the consequent of Eq. (3) does not cause false dismissal. We present the search algorithm using the matching windows in detail in Sect. 4.2.

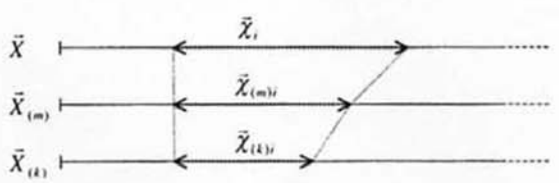


Fig. 4 Relationships among the window $\vec{\chi}_i$, m-moving average transformed window $\vec{\chi}_{(m)i}$, and matching window $\vec{\chi}_{(k)i}$.

4.1.1 Window Shifting

When the condition of non-overlapping windows in Eq. (4) does not hold, we solve the problem by window shifting defined in Definition 4:

Definition 4: For a window $\vec{\chi} = (\chi_i)$ $(0 \le i < n)$ of length n, the *window shifting* is the operation that adds a constant d_s to every component value χ_i of the window $\vec{\chi}$. The *shifted window* is denoted by $\vec{\chi} + d_s$. The value d_s is called the *shifting distance*. □

We shift up or down either $\vec{\chi}_i$ or $\vec{\tau}_j$ to make the condition in Eq. (4) satisfied and compute a new search range according to the shifting distance. We compute the shifting distance d_s using the minimum and maximum values of the windows $\vec{\chi}_i$ and $\vec{\tau}_j$. First, we define the difference values d_1 and d_2 for the windows $\vec{\chi}_i$ and $\vec{\tau}_j$ as follows:

$$\begin{aligned} d_1 &= \min(\vec{\chi}_i) - \max(\vec{\tau}_j), \\ d_2 &= \min(\vec{\tau}_j) - \max(\vec{\chi}_i) \end{aligned} \qquad (7)$$

where $\min(\omega)$ and $\max(\omega)$ are the minimum and maximum among the values constituting the window ω. If $d_1 \ge 0$ or $d_2 \ge 0$, i.e., the values in either window are all greater than the values in the other one, the condition of non-overlapping windows in Eq. (4) is satisfied, and thus, we perform the ordinary $\frac{\epsilon}{\sqrt{p}}$-range search. If $d_1 < 0$ and $d_2 < 0$, we must shift up or down either $\vec{\chi}_i$ or $\vec{\tau}_j$ to satisfy the condition.

Let the window $\vec{\chi}_i$ be the one to be shifted. As $\vec{\chi}_i$ is shifted away from the query window $\vec{\tau}_j$, the search range must increase to contain $\vec{\chi}_i$. This causes the false alarm to increase. Thus, we need to minimize the search range. The shifting distance d_s is computed as

$$d_s = \begin{cases} |d_1| & \text{if } |d_1| \le |d_2| \\ -|d_2| & \text{otherwise} \end{cases} \qquad (8)$$

Once the shifting distance d_s is determined, a new search range ϵ' is computed in such a way as to minimize false alarm while preventing false dismissal. We compute the search range ϵ' as

$$\epsilon' = \frac{\epsilon}{\sqrt{p}} + |d_s| \cdot \sqrt{w_m} \qquad (9)$$

where w_m is the length of the m-moving average transformed windows $\vec{\chi}_{(m)i}$ and $\vec{\tau}_{(m)j}$. Then, we construct the candidate set as those windows $\vec{\chi}_{(k)i}$ satisfying

$$d\left(\vec{\tau}_{(k)j}, \vec{\chi}_{(k)i} + d_s\right) \le \frac{\epsilon}{\sqrt{p}} + |d_s| \cdot \sqrt{w_m}. \qquad (10)$$

Theorem 2 shows that the search using the new search range ϵ' causes no false dismissal.

Theorem 2: The windows $\vec{\tau}_{(m)j}$ and $\vec{\chi}_{(m)i}$ are the m-moving average transformed results of the windows $\vec{\tau}_j$ and $\vec{\chi}_i$, respectively, and the windows $\vec{\tau}_{(k)j}$ and $\vec{\chi}_{(k)i}$ are their matching windows. The window $\vec{\chi}_{(k)i} + d_s$ is obtained by shifting the window $\vec{\chi}_{(k)i}$ by d_s. Then, the following relationship holds:

$$\begin{aligned} & d\left(\vec{\tau}_{(m)j}, \vec{\chi}_{(m)i}\right) \le \frac{\epsilon}{\sqrt{p}} \\ & \quad \Rightarrow d\left(\vec{\tau}_{(k)j}, \vec{\chi}_{(k)i} + d_s\right) \le \frac{\epsilon}{\sqrt{p}} + |d_s| \cdot \sqrt{w_m} \end{aligned} \qquad (11)$$

where the shifting distance d_s is computed using Eq. (8), and w_m is the length of the windows $\vec{\tau}_{(m)j}$ and $\vec{\chi}_{(m)i}$.

Proof: Refer to [14]. □

4.1.2 Using Multiple k-Indexes

The proposed algorithm can also use the k-indexes for more than one moving average order k to improve search performance. The more k-indexes are used, the better search performance is achieved. When the moving average m given at query time does not exist among the orders k of the k-indexes, i.e., there exists no k-index generated for the given order m, we choose one of the orders k using the following equation:

$$\kappa = \min\{k | k > m\}$$

We call the k-index for the chosen order κ the *κ-index*. When the condition of non-overlapping windows in Eq. (4) is satisfied, we construct the candidate set using κ-index and the following equation:

$$d\left(\vec{\chi}_{(\kappa)i}, \vec{\tau}_{(\kappa)j}\right) \le \frac{\epsilon}{\sqrt{p}}$$

When the condition is *not* satisfied, we use the following equation instead:

$$d\left(\vec{\tau}_{(\kappa)j}, \vec{\chi}_{(\kappa)i} + d_s\right) \le \frac{\epsilon}{\sqrt{p}} + |d_s| \cdot \sqrt{w_m}$$

Here, $w_m = w_\kappa + \kappa - m$ according to Eq. (6), where w_κ is the length of the matching window $\vec{\chi}_{(\kappa)i}$.

The proposed algorithm works well for the following reason. The execution time of the similar sequence matching algorithms such as the proposed one is dominated by accessing from the disk the data sequences in the candidate set obtained using the indexes [7]. Thus, the essential job to enhance the search performance is to keep the candidate set as small as possible.

We now show that the candidate set does not increase much for the proposed algorithm compared with the SUB94 algorithm [8]. The proposed algorithm has two differences from the SUB94 algorithm. First, when the condition of non-overlapping windows in Eq. (4) is satisfied, the proposed algorithm can perform range search even when the given moving average order m is not equal to the order k of the k-indexes. Second, when the condition is *not* satisfied, the proposed algorithm performs search by shifting the query windows. For the first case, the size of the candidate set increases as the given order m gets farther from k, i.e., as $|k-m|$ gets larger. However, we can keep the value $|k-m|$ within some boundary by using more k-indexes. For the second case, when the condition of non-overlapping windows in Eq. (4) is not satisfied, it means that the window $\vec{\chi}_i$ is close enough to the window $\vec{\tau}_j$, and thus, $\vec{\chi}_i$ is probably to be added to the candidate set. In contrast, the window $\vec{\chi}_i$ that is not likely to be added to the candidate set, i.e., that is not close enough to $\vec{\tau}_j$, satisfies the condition. Thus, even though we use the new search range ϵ' $(\geq \epsilon)$ for the windows that violate the condition of non-overlapping windows, the size of the candidate set does not increase much.

We show the results of performance evaluation through a series of experiments in Sect. 5.

4.2 Indexing and Searching Algorithms

In this subsection we explain detailed algorithms for generating the k-index and for searching using the k-index. The k-index is generated in almost the same way as in the plain search case discussed in Sect. 4.1, except that the minimum and maximum values are added to the information for each window. That is, for each sliding window $\vec{\chi}_{(k)i}$ of length w_k extracted from k-moving average transformed data sequence $\vec{S}_{(k)}$, we store f $(< w_k)$ coefficients obtained through DFT *plus* $\min(\vec{\chi}_i)$ and $\max(\vec{\chi}_i)$ values in an $(f+2)$-dimensional index structure.

The search algorithm using the k-index is presented in Fig. 5. For brevity, in the figure, we use the notation of windows $\vec{\tau}_{(k)}$ and $\vec{\chi}_{(k)}$ instead of their DFT transformed results $F(\vec{\tau}_{(k)})$ and $F(\vec{\chi}_{(k)})$. We explain only the case that the moving average order k of the k-index is not equal to the order m given at query time. First, the m-moving average transformed sequence $\vec{T}_{(m)}$ of the query sequence $\vec{T}$ are divided into p windows $\vec{\tau}_{(m)j}$ $(0 \leq j < p)$ as shown in Fig. 3. The length of the windows $\vec{\tau}_{(m)j}$ is $w_m = w_k + (k-m)$ according to Eq. (6), where w_k is the length of sliding windows stored in the k-index. For the matching window $\vec{\tau}_{(k)j}$ of each window $\vec{\tau}_{(m)j}$, the window search function in Fig. 5 is called. The function uses the matching window $\vec{\tau}_{(k)j}$ instead of the query window $\vec{\tau}_{(m)j}$ as an input, and performs range search to find the set containing all the windows $\vec{\chi}_{(m)i}$ that satisfy Eq. (2) using the k-index. The function operates on any kind of multidimensional index structure. The window searching is initiated by calling *RangeSearch*(*Root*, $\vec{\tau}_{(k)j}$, $\epsilon/\sqrt{p}$), where *Root* is the root node of the k-index. The candidate set is the union of all the windows $\vec{\chi}_{(k)i}$ returned by the function for each matching window $\vec{\tau}_{(k)j}$.

```
Function RangeSearch(Node N, Window $\vec{\tau}_{(k)}$, Range $\epsilon$)
returns a set of windows

// Passed parameters
Node N;                 // current node to be processed
Window $\vec{\tau}_{(k)}$;        // query window
Range $\epsilon$;              // search range

// Local variables
Set S;                  // set of windows (initially empty)

(1)  if N is a directory node then
(2)      for each directory entry E in N do
(3)          Calculate $d_1$ and $d_2$;
(4)          if $d_1 \geq 0$ or $d_2 \geq 0$ then
(5)              if $d(\vec{\tau}_{(k)}, M_E) \leq \epsilon$ then
                     $S = S \cup$ RangeSearch(E, $\vec{\tau}_{(k)}$, $\epsilon$);
(6)          else    // $d_1 < 0$ and $d_2 < 0$
(7)              Calculate $d_s$ and $\epsilon'$;
(8)              $M'_E = M_E + d_s$;
                     // shift $M_E$ by $d_s$ along every axis
(9)              if $d(\vec{\tau}_{(k)}, M'_E) \leq \epsilon'$ then
                     $S = S \cup$ RangeSearch(E, $\vec{\tau}_{(k)}$, $\epsilon$);
(10)         endif
(11)     end for
(12) else    // N is a data node
(13)     for each window $\vec{\chi}_{(k)}$ in N do
(14)         Calculate $d_1$ and $d_2$;
(15)         if $d_1 \geq 0$ or $d_2 \geq 0$ then
(16)             if $d(\vec{\tau}_{(k)}, \vec{\chi}_{(k)}) \leq \epsilon$ then Add $\vec{\chi}_{(k)}$ in S;
(17)         else    // $d_1 < 0$ and $d_2 < 0$
(18)             Calculate $d_s$ and $\epsilon'$;
(19)             $\vec{\chi}'_{(k)} = \vec{\chi}_{(k)} + d_s$;
(20)             if $d(\vec{\tau}_{(k)}, \vec{\chi}'_{(k)}) \leq \epsilon'$ then Add $\vec{\chi}_{(k)}$ in S;
(21)         endif
(22)     end for
(23) endif
(24) return S;
```

Fig. 5 Subsequence matching algorithm supporting moving average transform.

We now explain in detail each line of the *RangeSearch*() in Fig. 5. The lines (2)–(11) are for processing the directory nodes, and the lines (13)–(22) are for the data nodes. In lines (3) and (14), d_1 and d_2 are computed using Eq. (7). When in line (3), we need $\min(E)$ and $\max(E)$ values for the directory entry E instead of $\min(\vec{\chi}_i)$ and $\max(\vec{\chi}_i)$ in Eq. (7). The $\min(E)$ and $\max(E)$ are recursively defined as the minimum of $\min(SE_i)$ and the maximum of $\max(SE_i)$, respectively, where SE_i are the subentries of the entry E. The lines (4)–(5) and (15)–(16) perform ordinary ϵ-range search if $d_1 \geq 0$ or $d_2 \geq 0$. In line (5), M_E is the minimum bounding rectangle (MBR) for the entry E, and $d(\vec{\tau}_{(k)}, M_E)$ is the minimal Euclidean distance between $\vec{\tau}_{(k)}$ and M_E. The lines (7)–(9) and (18)–(20) are for processing the case where $d_1 < 0$ and $d_2 < 0$. The window shifting distance d_s and the new search range ϵ' is computed by Eqs. (8) and (9). In line (8), the MBR M'_E for the entry E is obtained by shifting the MBR M_E by d_s along every axis. That is, for MBR M_E that spans a range $[s_i, f_i]$ for each axis i $(0 \le i < f)$, MBR M'_E spans a range $[s_i+d_s, f_i+d_s]$. In lines (8) and (19), M'_E and $\vec{X}'_{(k)}$ can be easily obtained using the distributivity property of DFT [10]: $F(\vec{X}+b) = F(\vec{X})+bF(\vec{1})$, where $\vec{1}$ is a sequence consisting of only 1's. In line (9), $d\left(\vec{T}_{(k)}, M'_E\right)$ is the minimal Euclidean distance between the query window $\vec{T}_{(k)}$ and the MBR M'_E. If the MBR M'_E overlaps with the ϵ' range, the function is called recursively for the entry E. In line (20), if the shifted result of the window $\vec{X}_{(k)}$ is contained in the ϵ' range from the query window $\vec{T}_{(k)}$, the window $\vec{X}_{(k)}$ is added in the set S. In line (24), the final result set S of windows is returned.

5. Performance Evaluation

In this section we present the experimental results for performance evaluation of the proposed subsequence matching algorithm. We show that the performance for selectively-indexed case is comparable to that for fully-indexed case, and that the search performance gets better when more than one k-index is used. The search algorithm performed for the fully-indexed case is the one for the plain search case discussed in Sect. 4.1, which is a simple application of SUB94. We present the environment for experiments in Sect. 5.1 and the experimental results and analyses in Sect. 5.2.

5.1 Environment 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. To generate the query sequences $\vec{T}$, we have randomly chosen 128 out of 620 data sequences, and from them randomly extracted subsequences $\vec{Q} = (q_i)$ $(0 \le i < 256)$ of length 256 as in the reference [8]. We then have generated the query sequences $\vec{T} = (t_i)$ $(0 \le i < 256)$ by perturbing each q_i as in the reference [1] as follows:

$$t_i = q_i + z_i, \; z_i \in (-50, 50)$$

where z_i is an arbitrary value in the range $(-50, 50)$, and 50 is 5% of the average of $|q_{i+1} - q_i|$ $(0 \le i < 255)$ for all $\vec{Q}$. We set search ranges ϵ so that the final search result using ϵ should satisfy the selectivity defined below. We use the selectivity values 0.0001, 0.001, 0.01, and 0.1.

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

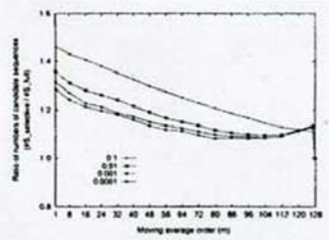
We have generated, for selectively-indexed case, the k-indexes for moving average orders $k = 64$ and 128, and, for fully-indexed case, ordinary indexes for the orders $m = 8i$ $(i = 1, 2, \ldots, 16)$ and $m = 1$, $k\pm1$. We have generated the k-indexes by adding only the minimum and maximum information for each window stored in the ordinary indexes, as explained in Sect. 4.2. Thus, the k-index is $(f+2)$-dimensional, while the ordinary index is f-dimensional. For the indexing, we use the coefficients of the first three frequencies obtained through DFT as in the reference [8]. The coefficients are all complex numbers, and we get two real numbers from the real and the imaginary parts of each coefficient. But, when the input data to DFT consists of only real numbers, the imaginary part of the first coefficient, which contains the largest energy, is always zero [15], [16]. Thus, we set the index dimensionality as $f = 5$. We set the length w_k of the sliding windows stored in the k-index as $w_k = 64$, and the length w_m of the sliding windows stored in the ordinary indexes for a moving average order m as $w_m = w_k + (k - m)$ using Eq. (6). We have used the R*-tree [4] as the multidimensional index structure to store sliding windows. Subtrails have been generated to contain multiple sliding windows as in the reference [8] and stored in the index structure. 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).

5.2 Experimental Results and Analyses

We have performed three experiments: The first and second experiments compare the numbers of subsequences in the candidate set and the elapsed times for algorithm execution between the selectively-indexed and fully-indexed cases. The third one compares the elapsed times for algorithm execution between the proposed and sequential scan algorithms.

The purpose of the first experiment is to show the amount of false alarm in the proposed algorithm compared with the algorithm by Faloutsos et al. [8]. Figure 6(a) shows the result using one k-index $(k = 128)$; Fig. 6(b) that using two k-indexes $(k = 64, 128)$. The horizontal axis represents the order m given at query time; the vertical axis the ratio of the number of subsequences for the selectively-indexed case, $\#S_{selective}$, divided by that for the fully-indexed case, $\#S_{full}$. Each value has been averaged for 128 queries.

In Fig. 6, the ratio increases as m gets farther from k. It is because the difference between the distances between the m-moving average transformed sequences and between the k-moving average transformed sequences increases as shown in Theorem 1. The ratio also increases a little bit as m gets closer to k. It is due to the new search range $\epsilon' = \frac{\epsilon}{\sqrt{p}} + \epsilon_{shift}$ resulting from the shifting of window $\vec{X}_{(k)i}$, where $\epsilon_{shift} = |d_s| \cdot \sqrt{w_m}$. Since the distance between the m-moving average transformed windows $\vec{T}_{(m)j}$ and $\vec{X}_{(m)i}$ decreases as m increases according to Theorem 1, the effect of ϵ_{shift} within ϵ' is exaggerated as m gets closer to k, and the false alarm due to ϵ_{shift} increases.


(a) Using one k-index ($k = 128$)

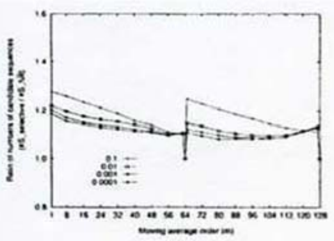
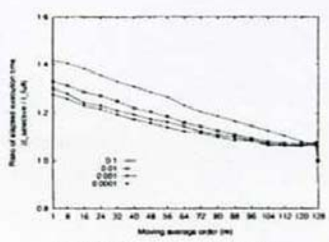

(b) Using two k-indexes ($k = 64$, 128)

Fig. 6 Ratio of the numbers of subsequences in the candidate sets ($\#S_{selective}/\#S_{full}$).


(a) Using one k-index ($k = 128$)

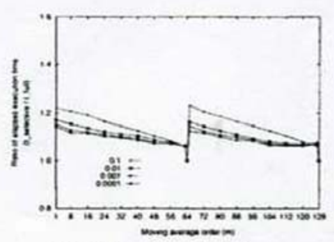
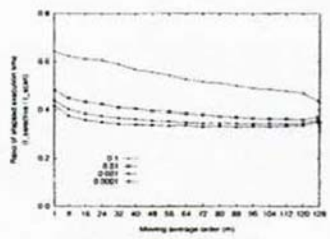

(b) Using two k-indexes ($k = 64$, 128)

Fig. 7 Ratio of the execution time for the selectively-indexed case divided by that of the fully-indexed case ($t_{selective}/t_{full}$).


(a) Using one k-index ($k = 128$)

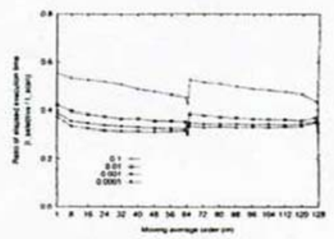

(b) Using two k-indexes ($k = 64$, 128)

Fig. 8 Ratio of the execution time of the proposed algorithm divided by that of the sequential scan algorithm ($t_{selective}/t_{scan}$).

If we use more k-indexes as in Fig. 6(b), we synchronize m with k again at the point a new k-index is created (say $m = 64$), and we can suppress overall increase of the ratios. As shown in Fig. 6, for cases that $m \leq 64$, we get better search performance using two k-indexes.

The second experiment compares the elapsed times for the algorithm execution for the selectively-indexed and fully-indexed cases. In many cases, the execution time of database programs is dominated by disk access time rather than CPU time. However, the disk access time varies depending on buffering and caching services by the OS. To measure the execution time consistently in this experiment, we make all the disk access routines bypass the buffering and caching services by the OS.

It is adequate to bypass the buffering and caching services of the OS for the following reasons. (1) Buffering and caching services help a query processor use the disk pages accessed by the previous queries at low costs, and therefore, their effects on search performance are highly dependent on the order of the queries. Thus, even though the same set of queries are processed, the search performance varies according to the order of the queries. (2) The proposed algorithm is executed only when user requests, and meanwhile other processes such as daemons may take memory buffer pages. Moreover, the proposed algorithm uses different k-indexes according to the moving average order m given at query time. Thus, the hit ratio for the proposed algorithm to find the required disk pages from the memory buffer is fairly low.

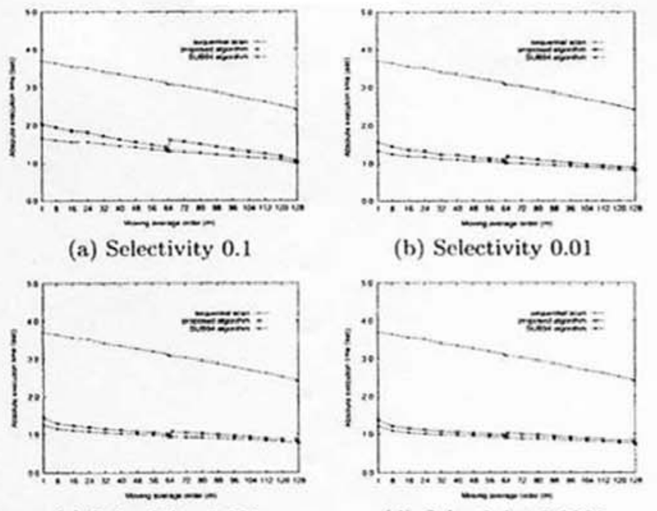
Figure 7(a) shows the result using one k-index ($k = 128$); Fig. 7(b) using two k-indexes ($k = 64$, 128). The vertical axis represents the ratio of the execution time for the selectively-indexed case, $t_{selective}$, divided by that for the fully-indexed case, t_{full}. Each value has been averaged for 128 queries. We can see that the trend is similar to that in Fig. 6. As in the first experiment, for cases that $m \leq 64$ where a new k-index is created in Fig. 7(b), we get better search performance than in Fig. 7(a).

Finally, the third experiment compares the elapsed times for the proposed algorithm and the sequential scan algorithm. As in the second experiment, we make all the disk access routines bypass the buffering and caching services. The sequential scan algorithm accesses all the data sequences directly from the disk and returns those that are within the given ϵ distance from the query sequence. In most cases, since the cost to access disks is very high, the sequential scan algorithm requires more execution time than the algorithms that access only the data sequences in the candidate set obtained using the indexes.

Figure 8(a) shows the result using one k-index ($k = 128$); and Fig. 8(b) using two k-indexes ($k = 64$, 128). The vertical axis represents the ratio of the execution time of the proposed algorithm, $t_{selective}$, divided by that of the sequential scan algorithm, t_{scan}. Each value has been averaged for 128 queries.

Figure 9 shows the absolute execution time of the proposed algorithm, SUB94 algorithm, and the sequential scan algorithm for selectivities 0.1, 0.01, 0.001, and 0.0001 when two k-indexes are used. The vertical axis represents the execution time in seconds. Each value has been averaged for 128 queries. In Fig. 9, the execution time decreases as the order m increases. It is because, as m increases, the length of the m-moving average transformed query sequence decreases as shown in Fig. 1, and so does the time to compute the Euclidean distance between the query sequence and the candidate subsequence.

We see that the proposed algorithm outperforms the sequential scan by up to 3.3 ($= 1 / 0.30$) times using two k-indexes, and that the search performance is more



(a) Selectivity 0.1 (b) Selectivity 0.01

(c) Selectivity 0.001 (d) Selectivity 0.0001

Fig. 9 Absolute execution time of the proposed algorithm, SUB94 algorithm, and the sequential scan algorithm when two k-indexes are used.

improved as the selectivity of the query gets smaller. In general, the queries with smaller selectivities are much more frequent than those with larger ones in database applications. This makes the proposed algorithm more useful in practical situations.

6. Conclusions

In this paper we have proposed a subsequence matching algorithm that supports moving average transform of arbitrary order that extends the existing subsequence matching algorithm (SUB94) proposed by Faloutsos et al. [8]. The simplest method to use SUB94 without any extension is to store and process the sliding windows extracted from the moving average transformed data sequences for every moving average order. In this case, however, we must generate one index for each moving average order, and will encounter serious overhead in storage space and data sequence insertion or deletion. We solve the problem using the notion of index interpolation. The proposed algorithm uses only one k-index for a pre-selected moving average order k and performs the subsequence matching for arbitrary order m ($\leq k$). The proposed algorithm can also use more than one k-index to improve search performance. We have proved that the proposed algorithm causes no false dismissal. We have shown by experiments that the search performance with only one k-index is comparable to that with the indexes for all the moving average orders, and that we get better search performance by using more k-indexes. We also have shown that the proposed algorithm outperforms the sequential scan algorithm significantly. The proposed algorithm has better search performance for queries with smaller selectivities and proves to be suitable for practical situations. The proposed algorithm can be used in various applications of moving average transform. Typical ones include analysis of stock price trends, estimation of product sales, and weather forecasting through temperature data analysis.

Acknowledgement

We thank Kyuseok Shim for providing incisive comments on earlier versions of this paper and for careful reading of the final version.

References

[1] R. Agrawal, C. Faloutsos, and A.N. Swami, "Efficient similarity search in sequence databases," Proc. Foundations of Data Organization and Algorithms, pp.69–84, Chicago, Illinois, Oct. 1993.

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

[3] R. Agrawal, G. Psaila, E.L. Wimmers, and M. Zait, "Querying shapes of histories," Proc. Int'l Conf. on Very Large Data Bases, pp.502–514, Zurich, Switzerland, Sept. 1995.

[4] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An efficient and robust access method for points and rectangles," Proc. Int'l Conf. on Management of Data, ACM SIGMOD, pp.322–331, Atlantic City, NJ, June 1990.

[5] S. Berchtold, D.A. Keim, and H.-P. Kriegel, "The X-tree: An index structure for high-dimensional data," Proc. Int'l Conf. on Very Large Data Bases, pp.28–39, Mumbai, India, Sept. 1996.

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

[7] K.-P. Chan and W.-C. Fu, "Efficient time series matching by wavelets," Proc. Int'l Conf. on Data Engineering, IEEE, pp.126–133, Sydney, Australia, March 1999.

[8] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos, "Fast subsequence matching in time-series databases," Proc. Int'l Conf. on Management of Data, ACM SIGMOD, pp.419–429, Minneapolis, Minnesota, June 1994.

[9] D.Q. Goldin and P.C. Kanellakis, "On similarity queries for time-series data: Constraint specification and implementation," Proc. Int'l Conf. on Principles and Practices of Constraint Programming, pp.137–153, Cassis, France, Sept. 1995.

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

[11] A. Guttman, "R-trees: A dynamic index structure for spatial searching," Proc. Int'l Conf. on Management of Data, ACM SIGMOD, pp.47–57, Boston, Massachusetts, June 1984.

[12] M. Kendall, Time-Series, 2nd Ed., Charles Griffin and Company, 1976.

[13] E. Kreyszig, Advanced Engineering Mathematics, 7th Ed., John Wiley & Sons, 1993.

[14] W.-K. Loh, S.-W. Kim, and K.-Y. Whang, Index Interpolation: A Subsequence Matching Algorithm Supporting Moving Average Transform of Arbitrary Order in Time-Series Databases, Technical Report, 99-11-001, Advanced Information Technology Research Center (AITrc), Taejon, Korea, Nov. 1999.

[15] A.V. Oppenheim and R.W. Schafer, Digital Signal Processing, Prentice-Hall, 1975.
[16] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery, Numerical Recipes in C—The Art of Scientific Computing, 2nd Ed., Cambridge University Press, 1992.
[17] D. Rafiei and A. Mendelzon, "Similarity-based queries for time series data," Proc. Int'l Conf. on Management of Data, ACM SIGMOD, pp.13–25, Tucson, Arizona, June 1997.
[18] T. Sellis, N. Roussopoulos, and C. Faloutsos, "The R^+-tree: A dynamic index for multidimensional objects," Proc. Int'l Conf. on Very Large Data Bases, pp.507–518, Brighton, England, Sept. 1987.
[19] R. Weber, H.-J. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces," Proc. Int'l Conf. on Very Large Data Bases, pp.194–205, New York, New York, Aug. 1998.
[20] B.-K. Yi, H.V. Jagadish, and C. Faloutsos, "Efficient retrieval of similar time sequences under time warping," Proc. Int'l Conf. on Data Engineering, IEEE, pp.201–208, Orlando, Florida, Feb. 1998.

Woong-Kee Loh received B.S. (1991) and M.S. (1993) degrees in Computer Science from Korea Advanced Institute of Science and Technology (KAIST). He is currently a Ph.D. candidate in Computer Science at KAIST. He has been a winter student of NHK Science and Technical Research Laboratories (STRL) in Tokyo, Japan in 1995. He has been a visiting student at Computer Science Department, Stanford University, California in 1997. He is a student member of the ACM and IEEE. His research interests include data mining/data warehousing, information retrieval, multimedia database systems, and multimedia content-based retrieval.

Sang-Wook Kim received the B.S. degree in Computer Engineering from Seoul National University in Korea at 1989, and earned the M.S. and Ph.D. degrees in Computer Science from Korea Advanced Institute of Science and Technology (KAIST) at 1991 and 1994, respectively. From 1994 to 1995, he worked with the Information and Electronics Research Center in Korea, as a Post-Doc. Since 1995, he has served as an Assistant Professor of the Division of Computer, Information, and Communications Engineering at Kangwon National University, Korea. Currently, he is working with the IBM T.J. Watson Research Center in Yorktown Heights, New York, as a visiting scientist. His research interests include data mining/data warehousing, multimedia information retrieval, transaction management, geographic information systems (GIS), and main memory databases. He is a member of the ACM and the IEEE.

Kyu-Young Whang graduated (Summa Cum Laude) from Seoul National University in 1973, and received the M.S. degrees from Korea Advanced Institute of Science and Technology (KAIST) in 1975, and Stanford University in 1982. He earned the Ph.D. degree from Stanford University in 1984. From 1983 to 1991, He was a Research Staff Member at the IBM T.J. Watson Research Center, Yorktown Heights, NY, where he performed various research projects in databases, office systems (including Office-by-Example), and expert systems. He is now a full professor at the Department of Computer Science and the Director of the Advanced Information Technology Research Center (AITrc) of KAIST. His research interests encompass data mining/data warehouses, database systems/storage systems, object-oriented databases, multimedia/hypermedia databases, geographic information systems (GIS), and digital libraries. He served as an IEEE Distinguished Visitor from 1989 to 1990, received the Best Paper Award from the 6th IEEE International Conference on Data Engineering, served the 5th IEEE International Conference on Data Engineering as a Program Co-Chair, and has served program committees of numerous international conferences including ACM SIGMOD and VLDB. He served the VLDB Conference as the Program Chair for Asia, Pacific, and Australia in 2000 and served the IFCIS CoopIS Conference as the Program Chair for Asia and Pacific Rim in 1998. He twice received the External Honor Recognition from IBM. He was an associate editor of the IEEE Data Engineering Bulletin from 1990 to 1993, and an editor of the Distributed and Parallel Database: An International Journal from 1991 to 1995. He is on the editorial boards of the VLDB Journal and International Journal of Geographic Information Systems. He is a Trustee of the VLDB Endowment and a Steering Committee Member of the DASFAA Conference. He is currently a vice president of Korea Information Science Society. He served the board of directors and was the Editor-in-Chief of the Journal of Korea information Science Society. He is a senior member of the IEEE and a member of the ACM.