Duality-Based Subsequence Matching in Time-Series Databases

Yang-Sae Moon, Kyu-Young Whang, and Woong-Kee Loh
Department of Computer Science and Advanced Information Technology Research Center (AITrc)
Korea Advanced Institute of Science and Technology (KAIST)
373-1, Kusong-Dong, Yusong-Gu, Taejon 305-701, Korea
{ysmoon,kywhang,woong}@mozart.kaist.ac.kr

Abstract

In this paper, we propose a new subsequence matching method, Dual Match, which exploits duality in constructing windows and significantly improves performance. Dual Match divides data sequences into disjoint windows and the query sequence into sliding windows, and thus, is a dual approach of the one by Faloutsos et al. (FRM in short), which divides data sequences into sliding windows and the query sequence into disjoint windows. We formally prove that our dual approach is correct, i.e., it incurs no false dismissal. We also prove that, given the minimum query length, there is a maximum bound of the window size to guarantee correctness of Dual Match and discuss the effect of the window size on performance. FRM causes a lot of false alarms (i.e., candidates that do not qualify) by storing minimum bounding rectangles rather than individual points representing windows to avoid excessive storage space required for the index. Dual Match solves this problem by directly storing points, but without incurring excessive storage overhead. Experimental results show that, in most cases, Dual Match provides large improvement in both false alarms and performance over FRM, given the same amount of storage space. In particular, for low selectivities (less than 10^{-4}*), Dual Match significantly improves performance up to 430-fold. On the other hand, for high selectivities (more than* 10^{-2}*), it shows a very minor degradation (less than 29%). For selectivities in between (*10^{-4}*~*10^{-2}*), Dual Match shows performance slightly better than that of FRM. Dual Match is also 4.10~25.6 times faster than FRM in building indexes of approximately the same size. Overall, these results indicate that our approach provides a new paradigm in subsequence matching that improves performance significantly in large database applications.*

1. Introduction

Time-series data are of growing importance in many new database applications such as data mining and data warehousing [10]. A time-series is a sequence of real numbers, representing values at specific time points. Typical examples of time-series data include stock prices, growth rates of companies, exchange rates, biomedical measurements, weather data, and etc. The time-series data stored in a database are called *data sequences*. Finding data sequences similar to the given *query sequence* from the database is called *similar sequence matching* [1, 5]. Owing to faster computing speed and larger storage devices, there has been a number of efforts to utilize the large amount of time-series data, and accordingly, similar sequence matching has become an important research topic in data mining [1, 2, 5, 8, 11].

Various similarity models have been studied in similar sequence matching. In this paper, we use the similarity model based on the Euclidean distance [1, 4, 5, 10]. Given two sequences $\vec{x} = \{x_1, x_2, ..., x_n\}$ and $\vec{y} = \{y_1, y_2, ..., y_n\}$ of the same length n, the Euclidean distance $D(\vec{x}, \vec{y})$ is defined as $\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$. We say two sequences $\vec{x}$ and $\vec{y}$ are *similar* if the distance $D(\vec{x}, \vec{y})$ is less than or equal to the user specified *tolerance* ϵ [1]. More specifically, we define that two sequences $\vec{x}$ and $\vec{y}$ are in ϵ-*match* if the distance between $\vec{x}$ and $\vec{y}$ is less than or equal to ϵ. We define *n-dimensional distance computation* as the operation that computes the distance between two sequences of length n.

Similar sequence matching can be classified into two categories [5]:

- *Whole matching*: Given N data sequences $S_1, ..., S_N$, a query sequence Q, and the tolerance ϵ, we find those data sequences that are in ϵ-match with Q. Here, the data and query sequences must have the same length.
- *Subsequence matching*: Given N data sequences S_1, ..., S_N of varying lengths, a query sequence Q, and the tolerance ϵ, we find all the sequences S_i, one or more subsequences of which are in ϵ-match with Q, and the offsets in S_i of those subsequences.

Thus, subsequence matching is a generalization of whole matching [4, 5, 16]. In this paper, we focus on subsequence matching.

Faloutsos et al. [5] have proposed a novel solution for subsequence matching on query sequences of varying lengths (we simply call this solution *FRM* by taking authors' initials). Subsequences similar to the query sequence can be found anywhere in a data sequence. In FRM, to find all possible subsequences, they use a *sliding window* of size ω starting from every possible offset in the data sequence. Then, they divide a query sequence into *disjoint*

1063-6382/01 $10.00 © 2001 IEEE

windows of size ω and retrieve similar subsequences by using those disjoint windows. They transform each sliding window to a point in a lower dimensional space (we call it *lower-dimensional transformation* or simply *transformation*). Since too many points are generated to be stored individually in an index, they construct *minimum bounding rectangles* (*MBRs*) that contain hundreds or thousands of points, using a heuristic method, and then, store those MBRs into a multidimensional index, R*-tree [3]. For subsequence matching, they first identify, using the index, those MBRs containing information to identify the subsequences, called *candidates*, that are potentially in ϵ-match with the query sequence. They subsequently refine the result by accessing the database and selecting only those subsequences that are in ϵ-match with the query sequence.

In this paper, we propose a new subsequence matching method, *Dual Match* (*Dual*ity-based subsequence *Match*ing), that reduces false alarms and improves performance significantly. We use the dual approach of FRM in constructing windows (we simply call it *duality*): i.e., we divide data sequences into disjoint windows and a query sequence into sliding windows. We formally prove that our dual approach is correct, i.e., it incurs no false dismissal. We also prove that, given the minimum length of the queries, there is a maximum bound of the window size to gurantee correctness of Dual Match and discuss the effect of the window size on performance.

FRM entails many *false alarms* (i.e., candidates that do not qualify) by storing only MBRs rather than individual points, and accordingly, degrades performance. In contrast, by dividing the data sequences into disjoint windows rather than sliding windows, Dual Match reduces the number of points to store drastically—to $1/\omega$ of that of FRM. Thus, Dual Match is able to store individual points instead of MBRs in the index. For subsequence matching, it first transforms the sliding windows of the query sequence into points, constructs range queries using these individual points and the user-specified tolerance ϵ, and then searches the index to get the candidates. By storing and searching with individual points directly, Dual Match reduces false alarms. Moreover, this method has an advantage of being faster in creating the index than FRM because it requires only $1/\omega$ lower-dimensional transformations of FRM.

The rest of this paper is organized as follows. Section 2 describes previous work. Section 3 explains the motivation of this research. Section 4 proposes Dual Match. Section 5 presents the results of performance evaluation. Section 6 summarizes and concludes the paper.

2. Related Work

We first summarize in Table 1 the notation to be used throughout the paper. The symbols in Table 1 are self explanatory and do not need further elaboration. We then review related work for whole matching and describe FRM, a representative research result for subsequence matching.

Table 1. Summary of notation.

Symbols	Definitions
$Len(S)$	Length of sequence S
Total_Len	Sum of lengths of all data sequences
$S[k]$	The k-th entry of sequence S ($1 \leq k \leq Len(S)$)
$S[i:j]$	Subsequence of S, including entries from the i-th one to the j-th
$S[i{:}k]S[k{+}1{:}j]$	$S[i:j]$ divided into $S[i:k]$ and $S[k+1:j]$
ω	Length of the sliding/disjoint window
$D(Q,S)$	Euclidean distance between sequences Q and S of equal length
ϵ	User-specified tolerance
s_i	The i-th disjoint window of sequence S

Whole Matching

Agrawal et al. [1] have first introduced a solution for similar sequence matching. The outline of the method is as follows. First, each data sequence of length n is transformed into the frequency domain by using Discrete Fourier Transform (DFT), and the first $f\,(\leq n)$ features are extracted. They are regarded as an f-dimensional point, and this point is indexed using the R*-tree [3]. Only a small number of features are extracted because of the difficulty in storing high-dimensional sequences in the R*-tree index due to dimensionality problem in multidimensional indexes (called *dimensionality curse* [13]). Next, a query sequence is similarly transformed to an f-dimensional point, and a range query constructed using the point and the given tolerance ϵ. Then, the R*-tree is searched to evaluate the query, a candidate set constructed consisting of the feature points that are in ϵ-match with the query sequence. This method guarantees no false dismissal (i.e., it does not miss a sequence in the result set), but may cause false alarms because it uses only f features instead of n. Thus, for each candidate sequence obtained, the actual data sequence is accessed from the disk; the distance from the query sequence computed; and the candidate is discarded if it is a false alarm. This last step, which eliminates false alarms, is called the *post-processing step* [1].

The function used for dimensionality reduction, such as extracting f features after DFT, is called the *feature extraction function* [5]. We have the following Lemma 1 for feature extraction functions.

Lemma 1 [5]: *To guarantee no false dismissals for range queries, the feature extraction function $F()$ must satisfy the following equation:*

$$D(F(S_1), F(S_2)) \leq D(S_1, S_2) \qquad (1)$$

DFT satisfies Lemma 1 [1, 5]. Recently, Chan and Fu [4] have proposed a similar sequence matching method that uses Haar Wavelet transform (we simply call it *Wavelet*) as the feature extraction function. They have shown that Wavelet also satisfies Lemma 1.

Subsequence Matching

Faloutsos et al.[5] have proposed the subsequence matching method (FRM) as a generalization of the whole matching method by Agrawal et al. [1]. They divide data sequences into sliding windows and a query sequence into disjoint windows. We explain FRM for three cases: 1) the length of the query sequence is equal to the window size, 2) the query sequence is composed of exactly $p\ (\geq 1)$ disjoint windows, and 3) the query sequence has a remainder when it is divided into p disjoint windows.

First, we explain the case where the query sequence has the length equal to that of a window. In this case, the problem becomes the one of finding windows that are in ϵ-match with the query sequence; thus, it can be solved using Agrawal et al.'s whole matching. That is, FRM transforms each sliding window to a point in the f-dimensional space. Next, it transforms the query sequence to a point in the f-dimensional space and makes a range query using the point and the tolerance ϵ. Lastly, it constructs a candidate set using the range query and discards false alarms through the post-processing step. FRM, however, generates almost $Total\_Len$ f-dimensional points corresponding to sliding windows for data sequences, and thus, needs f times more storage than is required by original data sequences. Moreover, the search performance may become even poorer than that of sequential scanning due to the excessive height of the R*-tree [5]. To solve this problem, FRM does not store individual points directly into the R*-tree, but stores only MBRs that contain hundreds or thousands of such points.

To construct MBRs, FRM uses heuristics in an attempt to minimize the number of disk accesses for the index. It first transforms a data sequence S into a trail consisting of $Len(S) - \omega + 1$ f-dimensional points. Next, it defines the marginal cost of a point using the estimated value (we call it the *estimated tolerance* ϵ') of 0.25[1] as the tolerance ϵ, and divides a trail into sub-trails using the cost [5]. FRM subsequently constructs an MBR for each sub-trail and stores it into the R*-tree with its starting and ending offsets in S and the identifier of S.

Next, we explain the case where the query sequence Q is composed of exactly p disjoint windows (i.e., $Len(Q) = p\omega$). FRM uses the following Lemma:

Lemma 2 [5]: *When two sequences S and Q of the same length are divided into p windows s_i and q_i $(1 \leq i \leq p)$ respectively, if S and Q are in ϵ-match, then at least one of the pairs (s_i, q_i) are in $\epsilon/\sqrt{p}$-match. That is, the following equation holds:*

$$D(S,Q) \leq \epsilon \implies \bigvee_{i=1}^{p} D(s_i, q_i) \leq \epsilon/\sqrt{p} \qquad (2)$$

Using Lemma 2, FRM divides the query sequence into p disjoint windows, transforms each window to an f-dimensional point, generates a range query using the point and $\epsilon/\sqrt{p}$, and then constructs a candidate set by searching the R*-tree. Since the candidates satisfy the necessary condition in Eq. (2), false dismissals do not occur.

Finally, we explain the case where the query sequence Q has a remainder when it is divided into p disjoint windows (i.e., $Len(Q) = p\omega + k, 1 \leq k \leq \omega - 1$). FRM uses the following Lemma:

Lemma 3 [5]: *If two sequences S and Q of the same length are in ϵ-match, then any pair of subsequences $(S[i:j], Q[i:j])$ are also in ϵ-match. That is, the following equation holds:*

$$D(S,Q) \leq \epsilon \implies D(S[i:j], Q[i:j]) \leq \epsilon \qquad (3)$$

According to Lemma 3, FRM does not cause any false dismissal by using the subsequence $Q[1:p\omega]$ instead of the query sequence $Q[1:p\omega + k]$. Since the subsequence $Q[1:p\omega]$ is composed of exactly p windows, FRM can find similar subsequences without false dismissals according to Lemma 2.

In summary, FRM works as follows. It first divides data sequences into sliding windows, transforms them into f-dimensional points, constructs the MBRs that contain multiple points, and stores them into the R*-tree. Next, it divides the query sequence into p disjoint windows, transforms each window to an f-dimensional point, makes a range query using the point and the tolerance $\epsilon/\sqrt{p}$, and constructs a candidate set by searching the R*-tree. Lastly, it performs the post-processing step to eliminate false alarms by accessing the data sequence and executing $Len(Q)$-dimensional distance computation for each candidate.

3. Motivation of the Research

In this section, we explain the motivation of our approach: in particular, why false alarms occur and how we reduce them. In similar sequence matching, the more false alarms occur, the more disk accesses and CPU operations for $Len(Q)$-dimensional distance computations are incurred in the post-processing step. Thus, false alarms are the main cause of performance degradation. In FRM, false alarms occur for the following three reasons: 1) use of feature extraction functions, 2) use of Lemmas 2 and 3, and 3) storing only MBRs in the index.

First, feature extraction functions cause false alarms because the lower-dimensional transformation is not distance-preserving. That is, although the distance between two f-

[1]FRM has used 0.25 for ϵ', the estimated tolerance to be given by the user, in the normalized domain space [0,1) of each axis.

dimensional points is less than or equal to ϵ, the actual distance between two ω-dimensional windows can be greater than ϵ. To reduce this kind of false alarms, one can increase the number of features used in the index or select a better feature extraction function. The recent Wavelet-based research of Chan and Fu [5] is a good example.

Second, using Lemmas 2 and 3 for long query sequences causes false alarms. That is, when two sequences S and Q are divided into p windows s_i and q_i $(1 \leq i \leq p)$ respectively, although a pair (s_i, q_i) are in $\epsilon/\sqrt{p}$-match, the distance between S and Q may be greater than ϵ. To reduce this kind of false alarms, we need to use as large windows as possible. For example, let the window size of the method A be twice as large as that of the method B. Then, by Lemma 2 or 3, a candidate subsequence of the method A must also be a candidate of the method B. However, the inverse does not hold. We define this effect the *window size effect*. The size of the window, however, must be less than or equal to the length of the query sequence; thus, the maximum window size is dependent on the length of the query sequence. In Section 4, we will explain this point in more detail when calculating the maximum window size that can be used for the proposed Dual Match.

Third, storing only MBRs instead of individual points causes false alarms. We explain this point using Figure 1. In Figure 1, P_i $(1 \leq i \leq 14)$ represents a point in the 2-dimensional space $(f = 2)$ to which a sliding window for a data sequence is transformed. The 14 P_i's are contained in an MBR. Q_1 and Q_2 represent the points for disjoint windows of a query sequence. In Figure 1, since Q_1 and Q_2 are in $\epsilon/\sqrt{p}$-match with the MBR, every P_i will be in the candidate set. In fact, however, no P_i is in $\epsilon/\sqrt{p}$-match with Q_1, and no P_i except P_8 and P_9 is with Q_2. Thus, we have many false alarms. We can reduce this kind of false alarms by storing every individual point of the MBR in the index.

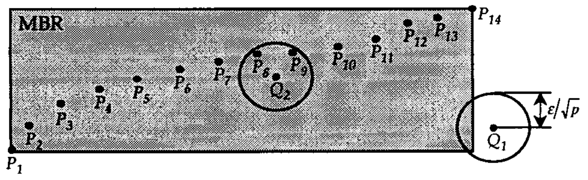


Figure 1. False alarms caused by storing only MBRs.

For example, in Figure 1, if every P_i were stored in the index, there would be no candidate for Q_1, but only two candidates P_8 and P_9 for Q_2. We define this effect the *point-filtering effect*. As we have explained in Section 2, however, if every individual point were stored in the index, then too much storage would be needed, and the performance degraded. Accordingly, in FRM, it is difficult to reduce the false alarms that are caused by the third reason.

In summary, the false alarms due to the first and second reasons are caused by the feature extraction function and the relative size of the query sequence compared with the window size. The false alarms due to the third reason, however, are caused by lack of the point-filtering effect. In Section 4, we introduce a subsequence matching method, Dual Match, that reduces the third type of false alarms fully utilizing the point-filtering effect.

4. Duality-based Subsequence Matching

The Concept

Dual Match divides data sequences into disjoint windows and the query sequence into sliding windows. This way, we are able to store and search individual points directly in the index without much storage overhead and improve disk and CPU performance.

We first define some terminology. Given a sequence S, a subsequence $S[i_2 : j_2]$ *includes* a subsequence $S[i_1 : j_1]$ if $i_1 \geq i_2$ and $j_1 \leq j_2$. When S is divided into fixed disjoint windows, we define the *included windows* for $S[i : j]$ as those disjoint windows included in $S[i : j]$. A subsequence of a specific length may have a different number of included windows depending on its position in S. For example, in Figure 2, the subsequence $S[i_1 : j_1]$ has one included window, but $S[i_2 : j_2]$ of the same length l has two. We define the *minimum number of included windows* for a subsequence of length l as the minimum one over all subsequences of the same length regardless of their positions in S. We can obtain this minimum using Lemma 4.

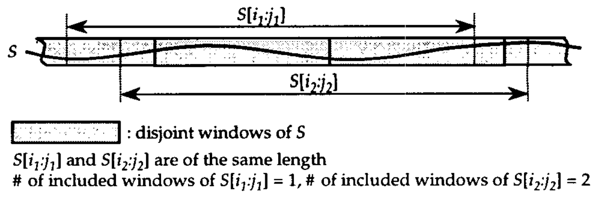

$S[i_1:j_1]$
S
$S[i_2:j_2]$
: disjoint windows of S
$S[i_1:j_1]$ and $S[i_2:j_2]$ are of the same length
# of included windows of $S[i_1:j_1]$ = 1, # of included windows of $S[i_2:j_2]$ = 2

Figure 2. Different numbers of included windows for two subsequences of the same length.

Lemma 4: *If the sequence S is divided into disjoint windows of size ω, the minimum number of included windows p for subsequences of length l is given by the following formula:*

$$p = \lfloor (l + 1)/\omega \rfloor - 1 \qquad (4)$$

PROOF: See the reference [9]. □

According to Lemma 4, a subsequence of length $Len(Q)$ includes at least $\lfloor (Len(Q) + 1)/\omega \rfloor - 1$ disjoint windows. We now derive Theorem 1, on which the correctness of Dual Match is based.

Theorem 1: *Suppose the data sequence S is divided into disjoint windows of size ω, and the query sequence Q into sliding windows of the same size ω. If the subsequence*

$S[i:j]$ of length $Len(Q)$ is in ϵ-match with Q, then at least one included window of $S[i:j]$ at a certain offset from $S[i]$ is in $\epsilon/\sqrt{p}$-match with the sliding window of Q at the same offset from $Q[1]$. Here, p is the minimum number of included windows for subsequences of length $Len(Q)$ given by Eq. (4).

PROOF: In Figure 3, suppose the subsequence $S[i:j]$ is in ϵ-match with the query sequence Q. $S[i:j]$ must include at least p disjoint windows $s_1, ..., s_p$, and also (possibly null) subsequences s_h (at the head) and s_t (at the tail). Thus, $S[i:j]$ can be represented as $s_h\, s_1 \cdots s_p\, s_t$. Similarly, Q can be represented as $q_h\, q_1 \cdots q_p\, q_t$ where $Len(q_h) = Len(s_h)$ and $Len(q_t) = Len(s_t)$. Then, we obtain Eq. (5) by using Lemmas 2 and 3.

$$D(S[i:j], Q) \le \epsilon \implies D(s_1 \cdots s_p, q_1 \cdots q_p) \le \epsilon \implies \bigvee_{k=1}^{p} D(s_k, q_k) \le \epsilon/\sqrt{p} \qquad (5)$$

Hence, if $S[i:j]$ and Q are in ϵ-match, at least one of p included windows of $S[i:j]$ (say s_k) must be in $\epsilon/\sqrt{p}$-match with a window q_k of Q. □

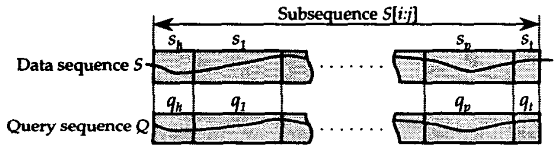


Figure 3. A subsequence $S[i:j]$ in ϵ-match with the query sequence Q.

At query time, since we use sliding windows and place them at every possible offset in the query sequence Q, the window q_k in Theorem 1 must be one of those sliding windows. According to Theorem 1, if we construct the candidate set with those subsequences that have an included window in $\epsilon/\sqrt{p}$-match with a sliding window of Q, i.e., that satisfy the necessary condition of Eq. (5), then we will not encounter any false dismissal.

Index Building Algorithm

Figure 4 shows the index building algorithm of Dual Match. The input to Algorithm BuildIndex is a database containing data sequences; the output an f-dimensional index, which will be used in subsequence matching. In Step 2.1 of the algorithm, we divide each data sequence into $\lfloor Len(S)/\omega \rfloor$ disjoint windows. The remaining subsequence $S[\lfloor \frac{Len(S)}{\omega} \rfloor * \omega + 1 : Len(S)]$, whose length is less than ω, is ignored by using Lemma 3. In Step 2.2.1, we transform a disjoint window to an f-dimensional point. In Step 2.2.2, we construct a record consisting of the transformed point *f-point*, the data sequence identifier *S-id*, and the start offset *dw-offset* of the disjoint window in S. The identifier will be used, when searching the index, to find the actual data sequence that contains the similar subsequence; the start offset to find the offset of the subsequence in the sequence. We subsequently insert the record into the index using the transformed point as the key.

```
Algorithm BuildIndex
Input:    Database db that contains data sequences
Output:   f-dimensional index that will be used for subsequence matching
Algorithm:
1 Initialize the index.
2 For each data sequence S with the identifier S-id in db, DO
  2.1 Divide S into ⌊Len(S)/ω⌋ disjoint windows.
  2.2 For each disjoint window with the start offset dw-offset, DO
      2.2.1 Transform the window to an f-dimensional point f-point by
            using the feature extraction function.
      2.2.2 Construct a record <f-point, S-id, dw-offset>.
      2.2.3 Insert the record, whose key is f-point, into the index.
```

Figure 4. The index building algorithm BuildIndex.

Dual Match has an important advantage: it is able to store the individual points, which have been transformed from disjoint windows, directly in the index without much storage overhead. It generates approximately $Total\_Len/\omega$ points by dividing data sequences into disjoint windows, and thus, the storage for the index is about f/ω of that for the original data sequences. This is only approximately $1/\omega$ of the storage that FRM would take if it stored (approximately $Total\_Len$) individual points directly in the index. In practice, since f is less than 10, and ω greater than 100 [4, 5], the storage for the index in Dual Match is less than $10\%\,(\frac{f}{\omega} = \frac{10}{100})$ of that for the original data sequences; the number of points stored in the index is less than $1\%\,(\frac{1}{\omega} = \frac{1}{100})$ of the sum of the lengths of all data sequences.

Dual Match has additional advantages: 1) it can use point access methods (PAMs) as the index, and 2) the index creation is very fast. Multidimensional index methods can be categorized into PAMs [12, 14, 15] that store points and spatial access methods (SAMs) [3, 7] that store spatial objects [6]. Since Dual Match stores points, it can use a PAM as the index with a flexibility of using various multidimensional indexes of differing characteristics. Dual Match can create the index much faster than FRM, since it needs only $1/\omega$ as many calls as in FRM to feature extraction functions, which constitute a major part of the CPU overhead.

Basic Subsequence Matching Algorithm

Figure 5 shows Basic Dual Match algorithm. The inputs to the algorithm are the time-series database, index, query sequence Q, and tolerance ϵ; the output is the set of sequences containing subsequences that are in ϵ-match with Q and offsets of those subsequences.

Algorithm Basic Dual Match consists of three steps: initialization, index searching, and post-processing. In the initialization step, we calculate the minimum number of

```
Algorithm Basic Dual Match
Input:  (1) Database db that contains data sequences
        (2) f-dimensional index that has been created by BuildIndex
        (3) Query sequence Q and tolerance ε
Output: Data sequences that contain subsequences that are in ε-match
        with Q and offsets of those subsequences
Algorithm:
1 Initialization
  1.1 Calculate the minimum number of included windows p.
  1.2 Divide Q into Len(Q) − ω + 1 sliding windows.
2 Index searching: for each sliding window Q[i:i +ω −1], DO
  2.1 Transform the window to an f-dimensional point.
  2.2 Construct a range query using the transformed point and $\epsilon/\sqrt{p}$.
  2.3 Search the index and include in the candidate set the records that
      are found together with the value i.
3 Post-processing: for each record <f-point, S-id, dw-offset> in the can-
  didate set, DO
  3.1 Read from db the candidate subsequence sub-S of the data sequence
      S. This is done using S-id. The offset of sub-S in S is calculated as
      'dw-offset − i + 1.' Here, i is the index of the sliding window that has
      been stored with this record in Step 2.3.
  3.2 If D(sub-S,Q) ≤ ε, then output S-id and the offset of sub-S.
```

Figure 5. The basic subsequence matching algorithm Basic Dual Match.

included windows $p = \lfloor (Len(Q)+1)/\omega \rfloor - 1$ for the subsequence of length $Len(Q)$ using Lemma 4, and divide the query sequence into $Len(Q) - \omega + 1$ sliding windows. In the index searching step, we construct the candidate set. We first transform each sliding window to an f-dimensional point and construct a range query using this point and $\epsilon/\sqrt{p}$. Next, we evaluate the range query, using the index, retrieving the qualifying points into the candidate set. In the post-processing step, for each record in the candidate set, we first read the candidate subsequence *sub-S* from the database in Step 3.1. If the sliding window is the i-th $(1 \leq i \leq Len(Q) - \omega + 1)$ one, then we calculate the start offset of *sub-S* in the data sequence S as '*dw-offset* − i + 1.' Here, *dw-offset* is the start offset in S of the disjoint window (point) in the candidate set. In Step 3.2, we remove false alarms keeping only those subsequences in ϵ-match with the query sequence. For each such subsequence *sub-S*, we output the identifier *S-id* of the data sequence S containing *sub-S* and the offset of *sub-S* in S.

Algorithm Basic Dual Match is very effective in reducing false alarms by using individual points rather than MBRs, i.e., by exploiting the point-filtering effect. However, it has a problem of evaluating many $(Len(Q) - \omega + 1)$ range queries—one for each sliding window. This could cause performance degradation. We present the Enhanced Dual Match algorithm to correct this problem.

Enhanced Subsequence Matching Algorithm

Rather than constructing a query for each point, Enhanced Dual Match constructs a query for an MBR that contains multiple points. This approach is similar to that of FRM, in which MBRs are constructed using multiple points for a data sequence. It is different in that it keeps the points in the MBR while FRM does not, and in that it uses MBRs for the query while FRM does for the data sequences. Since the search result for a sliding window of the query sequence may be similar to those for adjacent sliding windows, we use MBRs that contain multiple points for adjacent windows. Using MBRs to search the index tends to increase the size of the candidate set. Nevertheless, we can get the same candidate set as that of Basic Dual Match—despite the use of MBRs—by filtering false alarms in the index before accessing data sequences in the database. We do filtering by computing the f-dimensional distance between each point in the MBR and each point in the search result and by including in the candidate set only those points that are in $\epsilon/\sqrt{p}$-match. We define this filtering as *index-level filtering*. Index-level filtering is possible because we maintain all the points in an MBR. Figure 6 shows the algorithm Enhanced Dual Match. Like Basic Dual Match, Enhanced Dual Match consists of three steps: initialization, index searching, and post-processing.

```
Algorithm Enhanced Dual Match
Input:  (1) Database db that contains data sequences
        (2) f-dimensional index that has been created by BuildIndex
        (3) Query sequence Q and tolerance ε
Output: Data sequences that contain subsequences that are in ε-match
        with Q and offsets of those subsequences
Algorithm:
1 Initialization
  1.1 Calculate the minimum number of included windows p.
  1.2 Divide Q into Len(Q) − ω + 1 sliding windows and transform each
      window to an f-dimensional point.
  1.3 Construct MBRs using the transformed points.
2 Index searching: for each MBR, DO
  2.1 Construct a range query using the MBR and $\epsilon/\sqrt{p}$.
  2.2 Search the index using the range query and do index-level filtering
      (compute the distance between each point in the MBR and each
      point in the search result; include in the candidate set only the
      records having those points that are in $\epsilon/\sqrt{p}$-match together with
      the index i of the matching sliding window).
3 Post-processing: for each record <f-point, S-id, dw-offset> in the can-
  didate set, DO
  3.1 Read from db the candidate subsequence sub-S of the data sequence
      S. This is done using S-id. The offset of sub-S in S is calculated as
      'dw-offset − i + 1.' Here, i is the index of the sliding window that has
      been stored with this record in Step 2.2.
  3.2 If D(sub-S,Q) ≤ ε, then output S-id and the offset of sub-S.
```

Figure 6. The enhanced subsequence matching algorithm Enhanced Dual Match.

In the initialization step, we calculate the minimum number of included windows p, divide the query sequence into sliding windows, transform each sliding window to an f-dimensional point, and then construct MBRs that contain multiple points. We may use various techniques for constructing MBRs. Examples are 1) the heuristics used in FRM discussed in Section 2, 2) using a fixed number of points in an MBR, and 3) using only one MBR containing all the points. The detailed discussion, however, is not a focus of this paper and is left as a further study. In general, if the query sequence is long, using several MBRs is more effective since MBRs do not become too large. Experimental results for real stock data show that using 2~8 MBRs can improve the performance compared with using only one. In this paper, however, to simplify the problem, we use only one MBR.

In the index searching step, we construct the candidate set. We first make a range query using each MBR and the tolerance $\epsilon/\sqrt{p}$. Then, we retrieve the qualifying points by searching the index and construct the candidate set by using index-level filtering.

The post-processing step is the same as Basic Dual Match.

Maximum Window Size vs. Minimum Query Length

We explain the relationship between the maximum window size and the minimum length of a query sequence in Lemma 5 and discuss its implication.

Lemma 5: *If the minimum length of the query sequence is given by $Min(Q)$, then the maximum window size allowed in Dual Match is* $\lfloor(Min(Q)+1)/2\rfloor$.

PROOF: See the reference [9]. □

Given the same minimum length of the query sequence, the maximum window size of Dual Match is about half that of FRM because the former is $\lfloor(Min(Q)+1)/2\rfloor$ and the latter $Min(Q)$ [5]. As we have explained in Section 3, a smaller window causes more false alarms by the window size effect. Hence, the smaller maximum window size adds some tendency that Dual Match generates more false alarms than FRM. Nevertheless, Dual Match more than compensate for this effect by significantly reducing false alarms exploiting the point-filtering effect.

5. Performance Evaluation

Experimental Data and Environment

We have performed extensive experiments using three types of data sets. A data set consists of a long data sequence and has the same effect as the one consisting of multiple data sequences. The first data set, a real stock data set[2] used in FRM, consists of 329112 entries. We call this data set *STOCK-DATA*. The second data set, also used in FRM, contains random walk data consisting of five million entries. The data are generated synthetically: the first entry is set to 1.5, and subsequent entries are obtained by adding a random value in the range $(-0.001, 0.001)$ to the previous one. We call this data set *WALK-DATA*. The last data set contains pseudo periodic synthetic time-series data[3] consisting of one million entries. We call this data set *PERIODIC-DATA*. In PERIODIC-DATA, similar subsequences appear repeatedly with a long period. Changes among adjacent entries are small in STOCK-DATA and WALK-DATA; those in PERIODIC-DATA are relatively large.

All the experiments are conducted on a SUN Ultra 60 workstation with 512 Mbytes of main memory. To avoid the buffering effect of the UNIX file system and to guarantee actual disk I/Os, we use raw disks for data and index files. The page size for data and indexes is set to 4096 bytes. As the multidimensional index, we use R*-tree for both FRM and Dual Match. As the feature extraction function, we use the DFT and Wavelet transformations. We set the minimum length of the query sequence to be 512. Thus, the window size of FRM becomes 512, and that of Dual Match 256. We use 6 features[4], as has been done in FRM. We use 512, 768, and 1024 as the lengths of query sequences. They are uniformly distributed over various selectivities[5].

In FRM, the average number of points contained in an MBR varies depending on the estimated tolerance ϵ' used in the heuristics. This number, in turn, affects the number of false alarms and the size of the index. In the experiments, we make the index sizes and the storage requirements approximately the same—the difference is less than 10%—for fair comparison of the two methods. This is done by controlling ϵ' to make the number of points in an MBR for FRM and the number of entries in the disjoint window (window size) for Dual Match approximately the same and, in turn, to make the number of MBRs stored in FRM and the number of transformed points stored in Dual Match approximately the same. We further classify those experiments into two categories: 1) those using Wavelet (Case A) and 2) those using DFT (Case B). In addition, we also perform experiments for the case where the estimated tolerance ϵ' is 0.25, the same value used in the original experiments done in FRM (Case C).

For the experimental results, we measure the relative number of candidates, the relative number of page accesses[6], and the relative wall clock time of the two methods on a dedicated machine. We generate query sequences from the data sequences by taking subsequences of length $Len(Q)$ starting from random offsets [5]. To avoid effects of noise, we experiment with 10 different query sequences of the same length and use the average as the result. We perform experiments for selectivities in the range 10^{-6}~10^{-1} [5]. For STOCK-DATA, however, the minimum selectivity tested is approximately 3.0×10^{-6} since we have less than 329112 subsequences. We obtain the desired selectivity by controlling the tolerance ϵ for each query.

[2]This data set can be obtained from ftp://ftp.santafe.edu.

[3]This data set is one of those that are currently under construction with support from the National Science Foundation and can be obtained from http://kdd.ics.uci.edu/databases/synthetic/synthetic.html.

[4]With DFT, we have used the real part of the fourth complex number instead of the imaginary part of the first one, which is 0.

[5]$\text{Selectivity}(Q) = \frac{\text{\# of subsequences that are in } \epsilon\text{-match with } Q}{\text{\# of all data subsequences of length } Len(Q)}$

[6]# of page accesses = # of data page accesses + # of index page accesses

Experimental Results

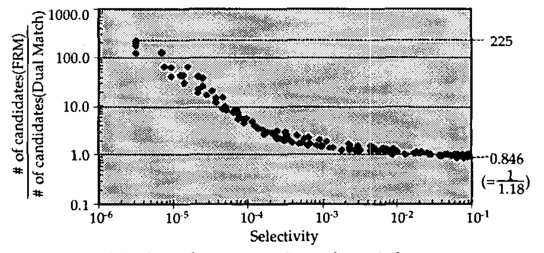
Here, we present the experimental results. We first explain in detail the results for Case A and then briefly mention those for Cases B and C.

1) STOCK-DATA: Figure 7 shows the experimental results using Wavelet for STOCK-DATA. Figure 7 (a) shows the relative number of candidates, Figure 7 (b) the relative number of page accesses, and Figure 7 (c) the relative wall clock time. In the figure, when the selectivity is less than 10^{-3}, Dual Match significantly reduces the number of candidates to as little as $\frac{1}{225}$ of that for FRM, reduces the number of page accesses by up to 4.49 times, and improves performance up to 10.1-fold. When the selectivity is in the range $10^{-3}\sim10^{-2}$, Dual Match shows performance slightly better than FRM in all three measures. On the other hand, when the selectivity is greater than 10^{-2}, Dual Match increases the number of candidates by up to 1.18 times, increases the number of page accesses by up to 1.23 times, and degrades performance by up to 1.21 times that of FRM. The increased number of candidates and performance degradation for higher selectivities are due to the window size effect; at the same time, the point-filtering effect is less eminent because the relative number of false alarms to the total number of candidates becomes smaller in higher selectivities.

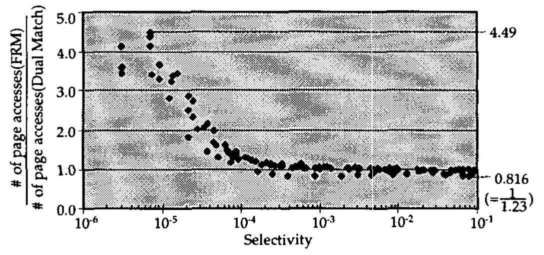
In Figure 7, the relative number of candidates is much higher than the relative number of page accesses and the relative wall clock time. The reason for this discrepancy is that adjacent subsequences are similar, and thus, can be accessed together being stored in the same data page. That is, if the subsequence $S[i : j]$ of the sequence S is similar to the query sequence Q, then many adjacent subsequences of $S[i : j]$, including $S[i-1 : j-1]$ and $S[i+1 : j+1]$, may very well be stored in the same data page. Compared to Dual Match, FRM accesses more (non-qualifying) adjacent subsequences included in the candidate set since many of them are represented together by one MBR in the index. Nevertheless, since those adjacent ones tend to be accessed together from the same data page, the relative number of I/O's—accordingly, the relative wall clock time—is smaller than the relative number of candidates.

2) WALK-DATA: The results using Wavelet for WALK-DATA show the same tendency as in Figure 7. We omit the detailed results of this experiment because of space limitation of the paper. See the reference [9] for the detailed result.

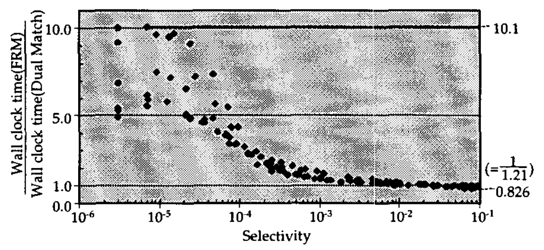
3) PERIODIC-DATA: Figure 8 shows the results using Wavelet for PERIODIC-DATA. Here, we have much larger improvement. When the selectivity is less than 10^{-4}, Dual Match drastically reduces the number of candidates to as little as $\frac{1}{8800}$ of that for FRM, reduces the number of page accesses by up to 26.9 times, and improves the performance up to 430-fold. PERIODIC-DATA has the characteristic that the changes among adjacent entries are relatively large. Accordingly, adjacent windows in PERIODIC-DATA tend to have distances among them larger than in STOCK-DATA. Thus, in FRM that stores MBRs of multiple adjacent windows, many windows far apart from one another can be included in the same MBR. Since these windows are included in the candidate set together, many false alarms are generated. In contrast, Dual Match does not cause this problem by storing individual points rather than MBRs. For this reason, PERIODIC-DATA shows larger relative number of candidates, relative number of page accesses, and relative wall clock time than STOCK-DATA does.



(a) The relative number of candidates



(b) The relative number of page accesses

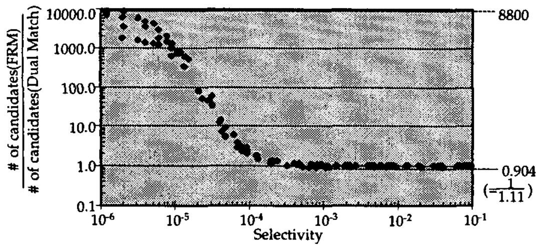


(c) The relative wall clock time

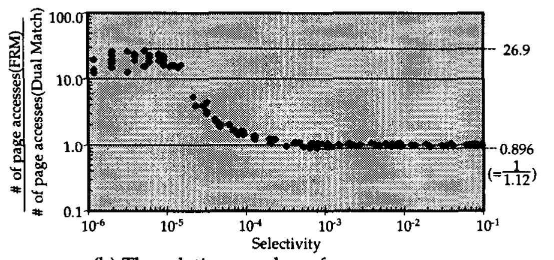
Figure 7. Performance comparison of Dual Match and FRM using Wavelet for STOCK-DATA.

The experimental results for Case B and Case C are similar to those for Case A. Table 2 summarizes the results for the three cases. In all three cases, Dual Match outperforms FRM significantly in lower selectivities with slight degradation in higher selectivities.

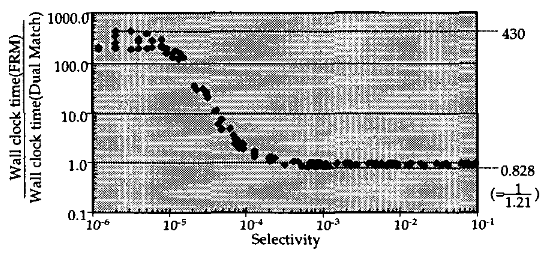
In summary, Dual Match drastically improves the performance over FRM due to the point-filtering effect for lower selectivities, but show slight degradation (less than 29%) for higher selectivities due to the window size effect. For very



(a) The relative number of candidates



(b) The relative number of page accesses



(c) The relative wall clock time

Figure 8. Performance comparison of Dual Match and FRM using Wavelet for PERIODIC-DATA.

large databases, which is typical in data mining, lower selectivities will be much more important than higher ones. Thus, Dual Match will be an effective tool for large database applications.

Table 3 shows the relative index creation time and storage space of the two methods. As shown in Table 3, Dual Match creates the index 4.10~25.6 times faster than FRM. The main reason for this speed up is that Dual Match needs only about $1/\omega$ lower-dimensional transformations of what FRM does. The storage space overheads of the two methods are almost the same for Cases A and B. For Case C, however, FRM needs about 5.19 times more storage space than Dual Match.

6. Conclusions

In this paper, we have proposed Dual Match, a new subsequence matching method based on duality in constructing windows. We have shown that Dual Match reduces false alarms and improves performance drastically compared with the previous method by Faloutsos et al. [5] (*FRM* in short). Dual Match divides data sequences into disjoint windows and the query sequence into sliding windows, and thus, is a dual approach of FRM, which divides data sequences into sliding windows and the query sequence into disjoint windows.

We have identified the reasons for false alarms: a major one is lack of the point-filtering effect in FRM. FRM stores in the index only MBRs instead of individual points to avoid excessive storage overhead, which would be f times as much as the size of the database itself. Here, each point corresponds to a *sliding* window of data sequences. Storing only MBRs, FRM cannot exploit the point-filtering effect. In contrast, Dual Match can store every individual point in the index without much storage overhead because the number of points to be stored in the index is only about $1/\omega$ as many as that of FRM. Here, each point corresponds to a *disjoint* window of data sequences. By storing individual points, Dual Match reduces false alarms drastically exploiting the point-filtering effect.

We have proven the correctness of Dual Match in Theorem 1, which guarantees that Dual Match perform subsequence matching without false dismissals. We also have derived the maximum allowable window size of Dual Match in Lemma 5. Given the same minimum length of the query sequence, the maximum window size of Dual Match is about half that of FRM. Since the smaller maximum window size causes more false alarms due to the window size effect, Dual Match shows performance slightly worse than that of FRM in higher selectivities.

Table 2. Experimental results of Dual Match and FRM for different lower-dimensional transformations and the estimated tolerances ϵ'.

(FRM/Dual Match, $10^{-6} \leq$ selectivity $\leq 10^{-1}$)

Experimental methods		STOCK	WALK	PERIODIC
Case A	# of candidates	0.846~225	0.789~178	0.904~8800
	# of page acc.	0.816~4.49	0.787~5.18	0.896~26.9
	wall clock time	0.826~10.1	0.788~14.4	0.828~430
Case B	# of candidates	0.859~379	0.798~320	0.886~3710
	# of page acc.	0.824~5.11	0.793~6.98	0.892~9.93
	wall clock time	0.964~6.84	0.748~14.5	0.771~41.8
Case C	# of candidates	1.02~876	0.748~39.0	0.868~8790
	# of page acc.	0.885~8.12	0.714~2.76	1.24~20.3
	wall clock time	0.965~33.5	0.788~3.99	0.860~255

Table 3. The relative index creation time and storage space of Dual Match and FRM.

(FRM/Dual Match)

Experimental methods	Index creation time			Average index storage space
	STOCK	WALK	PERIODIC	
Case A	7.60	4.10	4.89	1.03
Case B	12.0	6.14	6.06	1.10
Case C	7.16	5.32	25.6	5.19

We have performed extensive experiments using various types of data sets, feature extraction functions, and the estimated tolerances ϵ' (used in FRM). In most cases, Dual Match drastically reduces the number of candidates and improved performance. In particular, for lower selectivities (less than 10^{-4}), Dual Match reduces the number of candidates to as little as $\frac{1}{8800}$ of that for FRM, reduces the number of page accesses by up to 26.9 times, and improves performance up to 430-fold. For selectivities in between ($10^{-4} \sim 10^{-2}$), Dual Match shows performance slightly better than that of FRM. On the other hand, for higher selectivities (more than 10^{-2}), it shows a very minor degradation (less than 29%) by all three measures. This degradation is mainly due to the window size effect. In general, in large databases, users will require low selectivities to find only small number of similar subsequences. Thus, Dual Match will be an effective tool for large database applications.

Dual Match also provides excellent performance in index creation. Experimental results show that it is 4.10~25.6 times faster than FRM in building indexes of approximately the same size. We obtain this result because Dual Match requires only about $1/\omega$ of lower-dimensional transformations that FRM does.

Overall, these results indicate that our approach provides a new paradigm in subsequence matching that improves performance significantly in many variations and applications based on the FRM approach. Dual Match can also be used with newer types of transformations such as moving average transformation, shifting and scaling, and normalization. We are currently investigating into detailed issues as a further study.

Acknowledgements The authors benefited from visiting the Computer Science Department of Stanford University in Summers 1999 and 2000 in completing the work presented in this paper. This work was supported by the Korea Science and Engineering Foundation (KOSEF) through the Advanced Information Technology Research Center (AITrc).

References

[1] Agrawal, R., Faloutsos, C., and Swami, A., "Efficient Similarity Search in Sequence Databases," In *Proc. the 4th Int'l Conf. on Foundations of Data Organization and Algorithms*, Chicago, Illinois, pp. 69-84, Oct. 1993.

[2] Agrawal, R., Lin, K.-I., Sawhney, H. S., and Shim, K., "Fast Similarity Search in the Presence of Noise, Scaling, and Translation in Time-Series Databases," In *Proc. the 21st Int'l Conf. on Very Large Data Bases*, Zurich, Switzerland, pp. 490-501, Sept. 1995.

[3] Beckmann, N., Kriegel, H.-P., Schneider, R., and Seeger, B., "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, Atlantic City, New Jersey, pp. 322-331, May 1990.

[4] Chan, K.-P. and Fu, A. W.-C., "Efficient Time Series Matching by Wavelets," In *Proc. the 15th Int'l Conf. on Data Engineering*, Sydney, Australia, pp. 126-133, Feb. 1999.

[5] Faloutsos, C., Ranganathan, M., and Manolopoulos, Y., "Fast Subsequence Matching in Time-Series Databases," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, Minneapolis, Minnesota, pp. 419-429, May 1994.

[6] Gaede, V. and Guenther, O., "Multidimensional Access Methods," *ACM Computing Surveys*, Vol. 30, No. 2, pp. 170-231, June 1998.

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

[8] Jagadish, H. V., Mendelzon, A. O., and Milo, T., "Similarity-Based Queries," In *Proc. the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, San Jose, California, pp. 36-45, May 1995.

[9] Moon, Y.-S., Whang, K.-Y., and Loh, W.-K., Efficient Time-Series Subsequence Matching Using Duality in Constructing Windows, AITrc Technical Report 00-11-001, Advanced Information Technology Research Center (AITrc), KAIST, Taejeon, Korea, Jan. 2000, (http://aitrc.kaist.ac.kr/research/search.html).

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

[11] Rafiei, D., "On Similarity-Based Queries for Time Series Data," In *Proc. the 15th Int'l Conf. on Data Engineering*, Sydney, Australia, pp. 410-417, Feb. 1999.

[12] Seeger, B. and Kriegel, H.-P., "The Buddy-Tree: An Efficient and Robust Access Method for Spatial Data Base Systems," In *Proc. the 16th Int'l Conf. on Very Large Data Bases*, Brisbane, Queensland, Australia, pp. 590-601, Aug. 1990.

[13] Weber, R., Schek, H.-J., and Blott, S., "A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces," In *Proc. the 24th Int'l Conf. on Very Large Data Bases*, New York City, New York, pp. 194-205, Aug. 1998.

[14] Whang, K.-Y. and Krishnamurthy, R., Multilevel Grid Files, IBM Research Report RC11516, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, Nov. 1985.

[15] Whang, K.-Y., Kim, S.-W., and Wiederhold, G., "Dynamic Maintenance of Data Distribution for Selectivity Estimation," *The VLDB Journal*, Vol. 3, No. 1, pp. 29-51, Jan. 1994.

[16] Yi, B.-K., Jagadish, H. V., and Faloutsos, C., "Efficient Retrieval of Similar Time Sequences Under Time Warping," In *Proc. the 14th Int'l Conf. on Data Engineering*, Orlando, Florida, pp. 201-208, Feb. 1998.