PERGAMON

Information Systems 26 (2001) 279–293

www.elsevier.com/locate/infosys

Efficient time-series subsequence matching using duality in constructing windows☆

Yang-Sae Moon*, Kyu-Young Whang, Woong-Kee Loh

Department of Computer Science and Advanced Information Technology Research Center (AITrc), Korea Advanced Institute of Science and Technology (KAIST), Taejon 305-701, South Korea

Abstract

In this paper, we propose a new subsequence matching method, *Dual Match*. Dual Match 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. (Proceedings of the ACM SIGMOD International Conference on Management of Data, Seattle, Washington, 1994, pp. 419–429.) (*FRM* in short), which divides data sequences into sliding windows and the query sequence into disjoint windows. 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 save storage space for the index. Dual Match solves this problem by directly storing points without incurring excessive storage overhead. Experimental results show that, in most cases, Dual Match provides large improvement both in 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. Overall, these results indicate that our approach provides a new paradigm in subsequence matching that improves performance significantly in large database applications. © 2001 Elsevier Science Ltd. All rights reserved.

Keywords: Duality; Data mining; Subsequence matching; Time-series data; Similarity search

1. Introduction

A time-series is a sequence of real numbers, representing values at specific time points. Typical examples of time-series data include stock prices, exchange rates, and weather data. 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* [1,7]. 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 [1,6–9].

Various similarity models have been studied in similar sequence matching [1,2,10]. In this paper,

☆Recommended by Maurizio Lenzerini. This work was supported by the Korea Science and Engineering Foundation (KOSEF) through the Advanced Information Technology Research Center (AITrc).

*Corresponding author. Tel.: +82-42-869-5562; fax: +82-42-869-3510.

E-mail address: ysmoon@mzoart.kaist.ac.kr (Y.-S. Moon).

we use the similarity model based on the Euclidean distance [1,5,7,11]. In this model, we say that two sequences $X = \{X[1], \ldots, X[n]\}$ and $Y = \{Y[1], \ldots, Y[n]\}$ of the same length n are *similar* if the Euclidean distance $D(X, Y)$ ($= \sqrt{\sum_{i=1}^{n} (X[i] - Y[i])^2}$) is less than or equal to the user specified *tolerance* ε [1]. More specifically, we define that two sequences X and Y are in ε-*match* if $D(X, Y)$ is less than or equal to ε.

Similar sequence matching can be classified into two categories [7]:

- *Whole matching*: Given N data sequences $S_1, S_2, \ldots, S_N$, 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 $S_1, S_2, \ldots, S_N$ 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 [5–7,10]. In this paper, we focus on subsequence matching.

Faloutsos et al. [7] have proposed a novel solution for subsequence matching on query sequences of varying lengths (we simply call this solution *FRM* by taking authors' initials). In FRM, 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 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*) to avoid the high dimensionality problem [4,12] in multidimensional indexes. Since too many points are generated to be stored individually in an index, they construct *minimum bounding rectangles* (*MBRs*) that contain multiple points, 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.

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 this paper, we propose a new subsequence matching method, *Dual Match* (*Dua*-*l*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. 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, and thus, 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 individual points directly in the index, Dual Match reduces false alarms.

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

2. Related work

We 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.

2.1. Whole matching

Agrawal et al. [1] have introduced a solution for whole matching. The outline of the method is as follows. First, each data sequence of length n is

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 kth entry of sequence S ($1 \leqslant k \leqslant Len(S)$)
$S[i:j]$	Subsequence of S, including entries from the ith one to the jth (if $i > j$, then it means a null sequence of length 0)
$S[i:k]S[k+1:j]$	$S[i:j]$ divided into two subsequences $S[i:k]$ and $S[k+1:j]$
s_i	The ith disjoint window of sequence S ($= S[(i-1)*\omega+1 : i*\omega]$)

transformed to an f ($\leqslant n$)-dimensional point by using DFT (discrete fourier transform), and this point is indexed using the R*-tree [3]. Next, a query sequence is similarly transformed to an f-dimensional point, and a range query constructed using the point and the tolerance ε. Then, a candidate set is constructed by searching the R*-tree. Lastly, 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 is called the *feature extraction function* [7]. We have the following Lemma 1 for feature extraction functions.

Lemma 1 (Faloutsos et al. [7]). *To guarantee no false dismissals for range queries, the feature extraction function $F(\ )$ must satisfy the following equation*:

$$D(F(S), F(Q)) \leqslant D(S, Q). \tag{1}$$

All orthonormal transforms including DFT, DCT (discrete cosine transform), and most of the Wavelet transforms satisfy Lemma 1 [1]. Recently, Chan and Fu [5] have proposed a similar sequence matching method by using Haar Wavelet transform (we simply call it *Wavelet*).

2.2. Subsequence matching

Faloutsos et al. [7] have proposed the subsequence matching method (FRM) as a generalization of the whole matching method by Agrawal et al. [1]. We explain FRM for two algorithms: the index building and the subsequence matching algorithms.

In the index building algorithm, FRM divides data sequences into sliding windows. 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 [7]. 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 [7]. FRM subsequently constructs an MBR for each sub-trail and stores it into the R*-tree.

In the subsequence matching algorithm, FRM uses the following two Lemmas:

Lemma 2 (Faloutsos et al. [7]). *If two sequences S and Q of the same length are divided into p windows s_i and q_i ($1 \leqslant i \leqslant p$), respectively, then the following*

[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.

equation holds:

$$D(S,Q) \leqslant \varepsilon \Rightarrow \bigvee_{i=1}^{p} D(s_i, q_i) \leqslant \varepsilon/\sqrt{p}. \tag{2}$$

Lemma 3 (Faloutsos et al. [7]). *If* $S[i:j]$ *and* $Q[i:j]$ *are the subsequences of sequences* S *and* Q, *respectively, then the following equation holds*:

$$D(S,Q) \leqslant \varepsilon \Rightarrow D(S[i:j], Q[i:j]) \leqslant \varepsilon. \tag{3}$$

According to Lemmas 2 and 3, FRM divides the query sequence Q into $p(= \lfloor Len(Q)/\omega \rfloor)$ disjoint windows, transforms each window to an f-dimensional point, makes a range query using the point and the tolerance $\varepsilon/\sqrt{p}$, and constructs a candidate set by searching the R*-tree. Lastly, it performs the post-processing step to eliminate false alarms.

Since Lemmas 2 and 3 are used for long query sequences, there is the tendency that longer windows decrease false alarms, and we call this effect the *window size effect*. For example, let the window size of the method A be twice as large as that of the method B. Then, by Lemmas 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. Thus, to reduce false alarms, we need to use as large windows as possible. In Section 4.5, we will explain this point in more detail when calculating the maximum window size that can be used for the proposed Dual Match.

3. Motivation of the research

In this section, we explain the motivation of our approach. In similar sequence matching, the more false alarms occur, the more disk accesses and CPU operations for computing the $Len(Q)$-dimensional distance are incurred in the post-processing step. Thus, false alarms are the main cause of performance degradation.

We note that storing only MBRs instead of individual points is one of the main reasons for false alarms in FRM. We explain this point using Fig. 1. In Fig. 1, P_i $(1 \leqslant i \leqslant 14)$ represents a point in the 2-dimensional space ($f = 2$) to which a sliding window for a data sequence is transformed. The 14 P_is are contained in an MBR. Q_1 and Q_2 represent the points for disjoint windows of a query sequence. In Fig. 1, since Q_1 and Q_2 are in $\varepsilon/\sqrt{p}$-match with the MBR, every P_i will be in the candidate set. In fact, however, no P_i is in $\varepsilon/\sqrt{p}$-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. For example, in Fig. 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 would degrade. Accordingly, in FRM, it is difficult to reduce the false alarms that are caused by lack of the point-filtering effect. In Section 4, we introduce a subsequence matching method, Dual Match, that reduces this type of false alarms fully utilizing the point-filtering effect.

4. Dual Match: duality-based subsequence matching

4.1. 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 $i_1 \geqslant i_2$ and $j_1 \leqslant j_2$. 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 Fig. 2, the subsequence $S[i_1:j_1]$ has one included window, but $S[i_2:j_2]$ of the

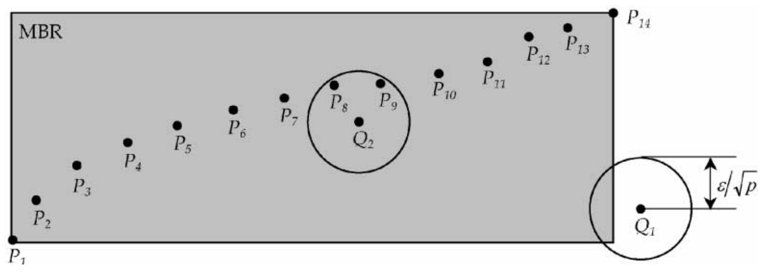

Fig. 1. False alarms caused by storing only MBRs.

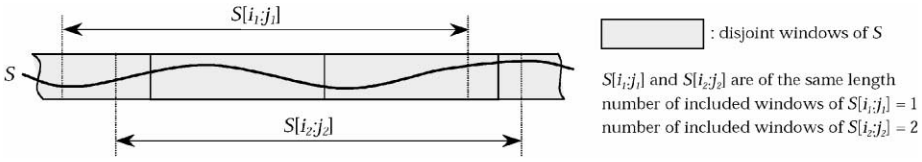
$S[i_1:j_1]$
S
$S[i_2:j_2]$
: disjoint windows of S
$S[i_1:j_1]$ and $S[i_2:j_2]$ are of the same length
number of included windows of $S[i_1:j_1]$ = 1
number of included windows of $S[i_2:j_2]$ = 2

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

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.

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*:

$$p = \lfloor (l+1)/\omega \rfloor - 1. \tag{4}$$

Proof. The minimum number of included windows for subsequences of length l is derived by neglecting the first $\omega - 1$ entries of the subsequences (the worst case, which minimizes the number of included windows) and taking the integral part of $(l - \omega + 1)/\omega$ for the remaining $l - (\omega + 1)$ entries. Thus, we obtain p as follows:

$$p = \lfloor (l - \omega + 1)/\omega \rfloor = \lfloor ((l+1) - \omega)/\omega \rfloor$$
$$= \lfloor (l+1)/\omega \rfloor - 1. \quad \square$$

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 validity 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 $\varepsilon/\sqrt{p}$-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 Fig. 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 $s_1, \ldots, s_p$, 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.

$$\begin{aligned} D(S[i:j], Q) \leqslant \varepsilon &\Rightarrow D(s_1 \cdots s_p, q_1 \cdots q_p) \leqslant \varepsilon \\ &\quad \text{(by Lemma 3)} \\ &\Rightarrow \bigvee_{k=1}^{p} D(s_k, q_k) \leqslant \varepsilon/\sqrt{p} \\ &\quad \text{(by Lemma 2).} \end{aligned} \tag{5}$$

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 $\varepsilon/\sqrt{p}$-match with a window q_k of Q. $\square$

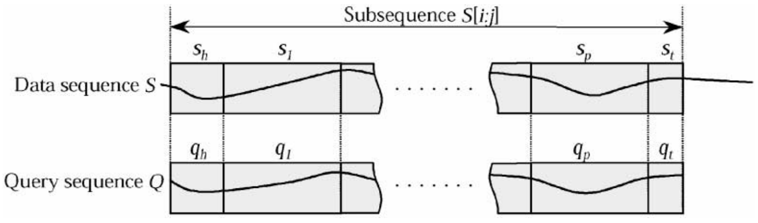

Fig. 3. A subsequence $S[i:j]$ in ε-match with the query sequence Q.

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 $\varepsilon/\sqrt{p}$-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.

4.2. Index building algorithm

In the index building algorithm, we first divide each data sequence into disjoint windows and transform each disjoint window to an f-dimensional point. We then construct a record consisting of the transformed point, the data sequence identifier, and the start offset of the disjoint window in S. We subsequently insert the record into the index using the transformed point as the key. We omit the detailed description of the algorithm since it is straightforward.

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 [5,7], the storage for the index in Dual Match is less than 10% ($f/\omega = 10/100$) of that for the original data sequences; the number of points stored in the index is less than 1% ($1/\omega = 1/100$) of the sum of the lengths of all data sequences.

Dual Match has an additional advantage: it can use point access methods (PAMs) as the index. Multidimensional index methods can be categorized into PAMs that store points and spatial access methods (SAMs) that store spatial objects [13]. Since Dual Match stores points, it can use a PAM as the index with a flexibility of using various multidimensional indexes of differing characteristics.

4.3. Basic subsequence matching algorithm

In the basic subsequence matching algorithm, we first calculate the minimum number of 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. We then transform each sliding window to an f-dimensional point and construct a range query using this point and $\varepsilon/\sqrt{p}$. Next, we evaluate the range query, using the index, retrieving the qualifying points into the candidate set. Lastly, we perform the post-processing step, i.e., for each record in the candidate set, we first read the candidate subsequence from the database, and then, remove false alarms keeping only those subsequences in ε-match with the query sequence. We also omit the detailed description of the algorithm since it is straightforward.

The basic algorithm 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 subsequence matching algorithm to correct this problem.

4.4. Enhanced subsequence matching algorithm

Rather than constructing a query for each point, the enhanced subsequence matching algorithm 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 the basic algorithm—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 $\varepsilon/\sqrt{p}$-match. We define this filtering as *index-level filtering*. Index-level filtering is possible because we maintain all the points in an MBR. Fig. 4 shows Enhanced Dual Match algorithm. Algorithm Enhanced Dual Match consists of three steps: initialization, index searching, and post-processing.

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. In Section 5, we will discuss this point in more detail.

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

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 ith $(1 \leqslant i \leqslant Len(Q) - \omega + 1)$ 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 of the data sequence S containing *sub-S* and the offset of *sub-S* in S.

4.5. 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 Q_{min}, then the maximum window size allowed in Dual Match is $\lfloor (Q_{min}+1)/2 \rfloor$.*

Proof. By Theorem 1, the minimum number of included windows for subsequences of length Q_{min} must be equal to or greater than one. Thus, we obtain the relationship between the minimum length of a query sequence and the window size as follows:

$$p = \lfloor (Q_{min}+1)/\omega \rfloor - 1 \geqslant 1$$
$$\Leftrightarrow (Q_{min}+1)/\omega \geqslant 2 \Leftrightarrow Q_{min} \geqslant 2\omega - 1.$$

Since ω is an integer, the maximum window size is $\lfloor (Q_{min}+1)/2 \rfloor$. $\square$

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 $\lfloor (Q_{min}+1)/2 \rfloor$ and the latter is Q_{min} [7]. As we have explained in Section 3, a smaller window causes more false alarms by the window

```
Algorithm Enhanced Dual Match
Input:  (1) Database db that contains data sequences
        (2) f-dimensional index that has been created by the index building algorithm
        (3) Query sequence Q and tolerance $\varepsilon$
Output: Data sequences that contain subsequences that are in $\varepsilon$-match with Q and offsets of those
        subsequences
Algorithm:
1 Initialization
  1.1 Calculate the minimum number of included windows p as $\lfloor (Len(Q)+1)/\omega \rfloor - 1$.
  1.2 Divide Q into 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 $\varepsilon/\sqrt{p}$ .
  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 $\varepsilon/\sqrt{p}$-match together with the index i of
      the matching sliding window).
3 Post-processing: for each record <the transformed point, the data sequence identifier S-id, the
  start offset dw-offset of the disjoint window> in the candidate 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 sub-S and Q are in $\varepsilon$-match, then output S-id and the offset of sub-S.
```

Fig. 4. The enhanced subsequence matching algorithm Enhanced Dual Match.

size effect. Hence, the smaller maximum window size adds some tendency that Dual Match generates more false alarms than FRM. Nevertheless, Dual Match compensates for this effect by significantly reducing false alarms exploiting the point-filtering effect.

5. Performance evaluation

5.1. Experimental data and environment

To prove the effectiveness of Dual Match, 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 [7], 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

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

[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.

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 [3] for both FRM [7] 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.

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 [7] (Case C).

For the experimental results, we measure the relative number of candidates, the relative number of page accesses,[5] 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 [7]. To avoid effects of noise, we experiment with 10 different query sequences of the same length and use the average as the result. We define the selectivity of a query as in Eq. (6). We perform experiments for selectivities in the range 10^{-6}–10^{-1} as has been done in FRM [7]. For STOCK-DATA, however, the minimum selectivity tested is approximately 3.0×10^{-6} since we have less than 329,112 subsequences. We obtain the desired selectivity by controlling the tolerance ε for each query.

$$Selectivity(Q) = \frac{\text{the number of subsequences that are in } \varepsilon\text{-match with the query sequence } Q}{\text{the number of all possible subsequences of length } Len(Q) \text{ in the database}}. \qquad (6)$$

5.2. Experimental results

First, to find the optimal number of range queries used in the enhanced subsequence matching algorithm, we have performed experiments to measure the index searching (including index-level filtering) time by using STOCK-DATA. Figs. 5(a) and (b) show the index searching time for the query sequences of lengths 512 and 1024, respectively. As shown in the figures, in the cases where the numbers of range queries are greater than 16, performance degrades compared with the other cases. On the other hand, in the cases where the numbers are less than 16, there is only minor difference in performance. That is, if the numbers are greater than 16, the index searching time increases due to multiple range queries. On the other hand, if they are less than 16, the results are

[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. Every coefficient at the end is the complex conjugate of a coefficient at the beginning and is as strong as its counterpart due to the symmetry property of DFT for real-valued sequences [14]. Explaining in more detail, the ith $(2 \leqslant i \leqslant \omega)$ complex number is the complex conjugate of the $(\omega - i + 2)$th one. Thus, we extract features by multiplying the second $\sim$ fourth complex numbers by $\sqrt{2}$ when we use DFT as a feature extraction function. This reflects the same effect as Rafiei and Mendelzon's approach [14], which multiplies $\sqrt{2}$ rather than 2 to construct the region of a range query.

[5] The number of page accesses = the number of data page accesses + the number of index page accesses.

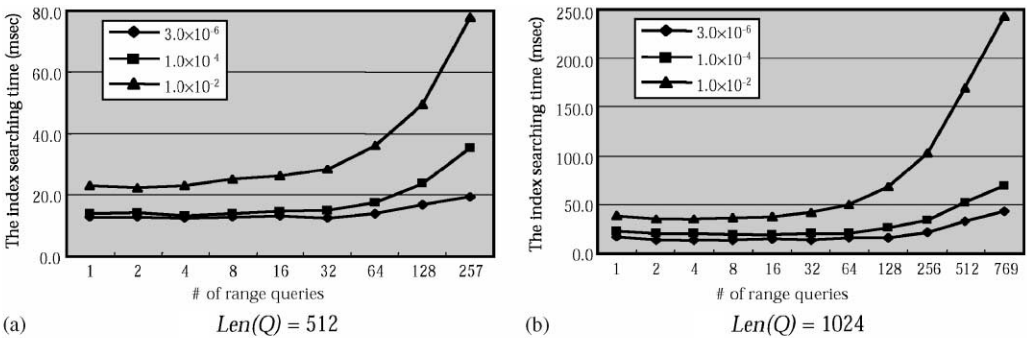

Fig. 5. The index searching time for varying numbers of range queries.

similar due to the balance between the effect of the size of the MBR and that of the number of range queries. In this paper, to simplify the problem, we use only one MBR.

Now, we present the experimental results of Dual Match and FRM. We first explain in detail the results for Case A and then briefly mention those for Cases B and C.

(1) STOCK-DATA: Fig. 6 shows the experimental results using Wavelet for STOCK-DATA. Fig. 6(a) shows the relative number of candidates, Fig. 6(b) the relative number of page accesses, and Fig. 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 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}–10^{-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 Fig. 6, 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.

Fig. 7(a) shows the relative number of index page accesses of the two methods, and Fig. 7(b) the relative number of data page accesses. As we see in Fig. 7, most of the enhancement in disk page accesses of Dual Match is due to improvement of data page accesses. Index page access performance in Dual Match is slightly better than in FRM. All the other experiments show the similar tendency as in Fig. 7.

(2) WALK-DATA: Fig. 8 shows the results using Wavelet for WALK-DATA. They show the same tendency as in Fig. 6. In particular, when the selectivity is less than 10^{-4}, Dual Match reduces the number of candidates to as little as 1/178 of that for FRM, reduces the number of page accesses by up to 5.18 times, and improves performance up to 14.4-fold. When the selectivity

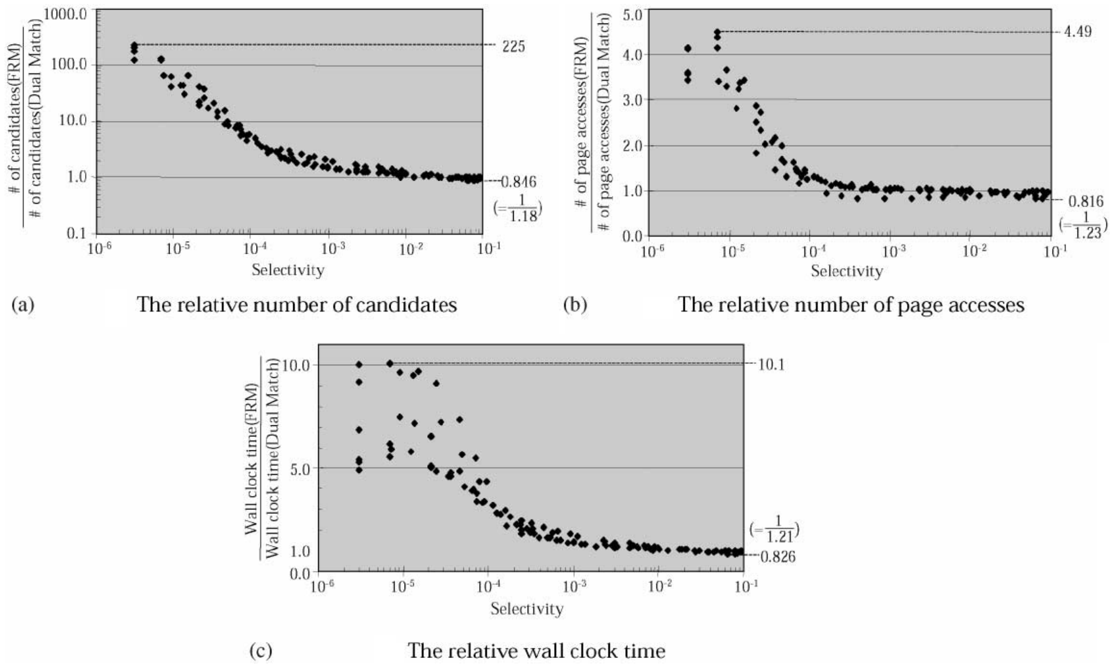

Fig. 6. Performance comparison of Dual Match and FRM using Wavelet for STOCK-DATA.

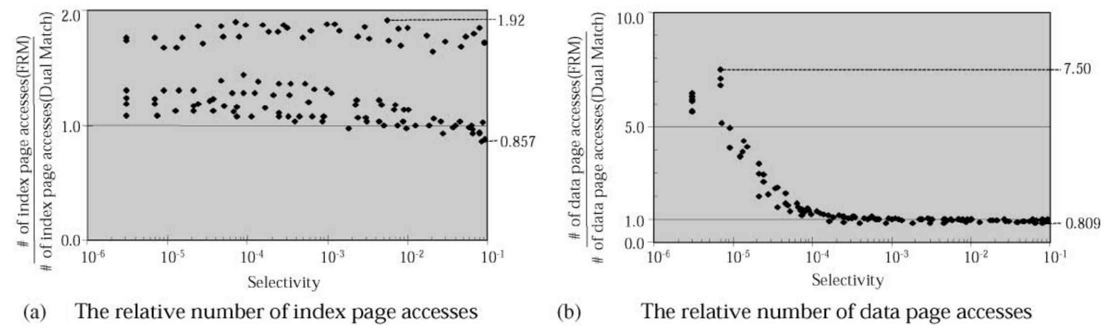

Fig. 7. Comparison of index and data page accesses of Dual Match and FRM using Wavelet for STOCK-DATA.

is greater than 10^{-2}, however, Dual Match increases the number of candidates by up to 1.27 times (21% in terms of relative number of candidates), increases the number of page accesses by up to 1.27 times (21% in terms of relative number of page accesses), and slightly degrades the performance by up to 1.27 times (21% in terms of relative wall clock time) that of FRM.

(3) PERIODIC-DATA: Fig. 9 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 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-

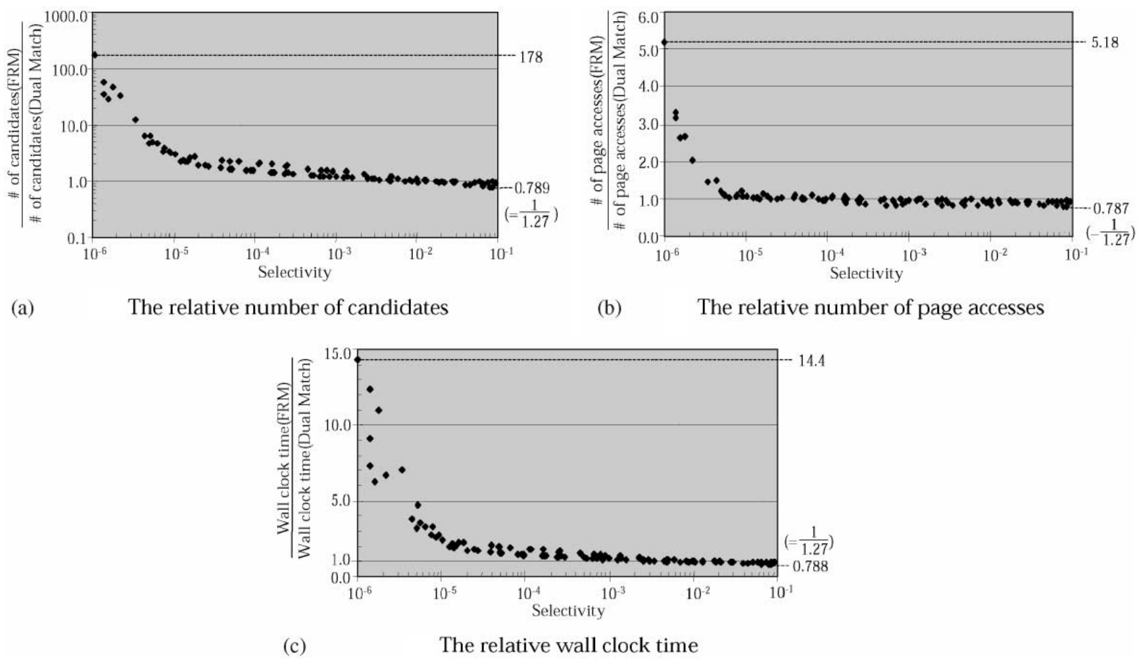

Fig. 8. Performance comparison of Dual Match and FRM using Wavelet for WALK-DATA.

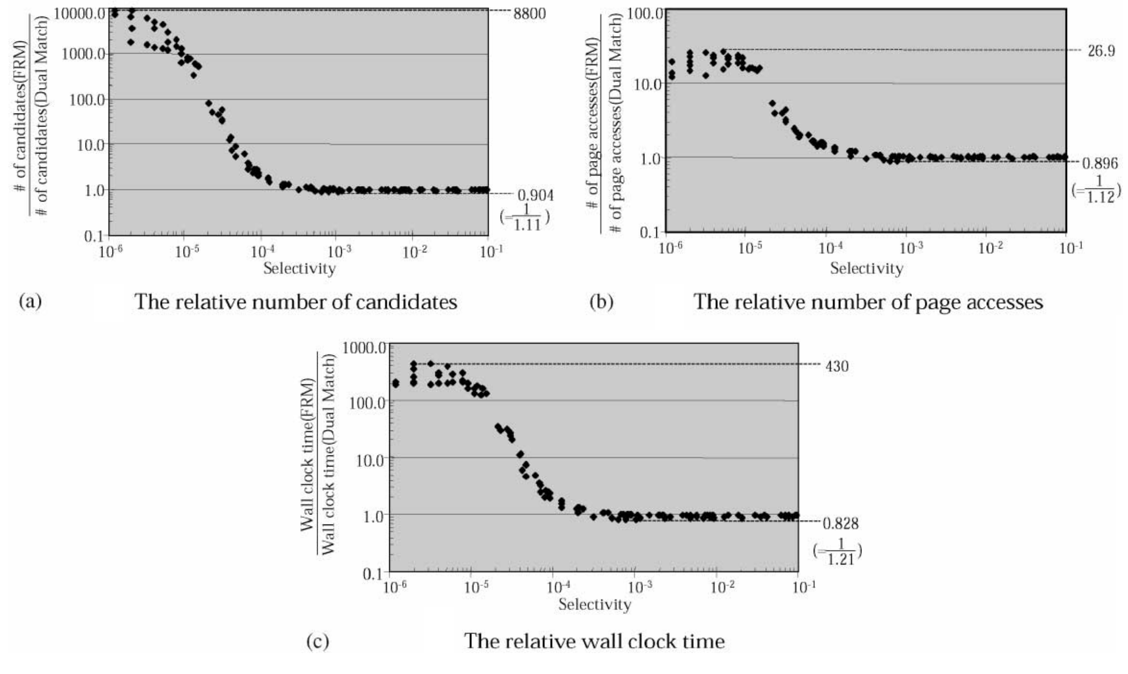

Fig. 9. Performance comparison of Dual Match and FRM using Wavelet for PERIODIC-DATA.

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

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

Experimental methods			STOCK-DATA	WALK-DATA	PERIODIC-DATA
Case A	Wavelet	# Of candidates	0.846–225	0.789–178	0.904–8800
	(# of points $\approx$	# Of page accesses	0.816–4.49	0.787–5.18	0.896–26.9
	# of MBRs)	Wall clock time	0.826–10.1	0.788–14.4	0.828–430
Case B	DFT	# Of candidates	0.859–379	0.798–320	0.886–3710
	(# of points $\approx$	# Of page accesses	0.824–5.11	0.793–6.98	0.892–9.93
	# of MBRs)	Wall clock time	0.964–6.84	0.748–14.5	0.771–41.8
Case C	Wavelet	# Of candidates	1.02–876	0.748–39.0	0.868–8790
	($\varepsilon' = 0.25$)	# Of page accesses	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

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 or WALK-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 show larger relative number of candidates, relative number of page accesses, and relative wall clock time than STOCK-DATA or WALK-DATA do. This is also the reason why those measures for STOCK-DATA are somewhat larger than those for WALK-DATA:[6] the average change between adjacent entries in WALK-DATA is ± 0.0005, but that in STOCK-DATA is ± 0.0008.

The experimental results for Cases B and 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 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.

[6] See the case where selectivity $= 10^{-5}$, for example.

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. [7] (*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 noted that one of the major reasons for false alarms 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 validity 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.

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 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}–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. 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.

Acknowledgements

We would like to thank Byoung-Yong Moon for helping in revising an earlier English version of this paper.

References

[1] R. Agrawal, C. Faloutsos, A. Swami, Efficient similarity search in sequence databases, Proceedings of the fourth International Conference on Foundations of Data Organization and Algorithms, Chicago, Illinois, 1993, pp. 69–84.

[2] R. Agrawal, K.-I. Lin, H. S. Sawhney, K. Shim, Fast similarity search in the presence of noise, scaling, and translation in time-series databases, Proceedings of the 21st International Conference on Very Large Data Bases, Zurich, Switzerland, 1995, pp. 490–501.

[3] N. Beckmann, H.-P. Kriegel, R. Schneider, B. Seeger, The r*-tree: an efficient and robust access method for points and rectangles, Proceedings of the ACM SIGMOD International Conference on Management of Data, Atlantic City, New Jersey, 1990, pp. 322–331.

[4] S. Berchtold, C. Bohm, H.-P. Kriegel, The pyramid-technique: towards breaking the curse of dimensionality, Proceedings of the ACM SIGMOD International Conference on Management of Data, Seattle, Washington, 1998, pp. 142–153.

[5] K.-P. Chan, A.W.-C. Fu, Efficient time series matching by wavelets, Proceedings of the 15th IEEE International Conference on Data Engineering, Sydney, Australia, 1999, pp. 126–133.

[6] K.W. Chu, M.H. Wong, Fast time-series searching with scaling and shifting, Proceedings of the 15th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Philadelphia, Pennsylvania, 1999, pp. 237–248.

[7] C. Faloutsos, M. Ranganathan, Y. Manolopoulos, Fast subsequence matching in time-series databases, Proceedings of the ACM SIGMOD International Conference on Management of Data, Seattle, Washington, 1994, pp. 419–429.

[8] H.V. Jagadish, A.O. Mendelzon, T. Milo, Similarity-based queries, Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, San Jose, California, 1995, pp. 36–45.

[9] D. Rafiei, On similarity-based queries for time series data, Proceedings of the IEEE 15th International Conference on Data Engineering, Sydney, Australia, 1999, pp. 410–417.

[10] B.-K. Yi, H.V. Jagadish, C. Faloutsos, Efficient retrieval of similar time sequences under time warping, Proceedings of the 14th IEEE International Conference on Data Engineering, Orlando, Florida, 1998, pp. 201–208.

[11] D. Rafiei, A. Mendelzon, Similarity-based queries for time series data, Proceedings of the ACM SIGMOD International Conference on Management of Data, Tucson, Arizona, 1997, pp. 13–25.

[12] R. Weber, H.-J. Schek, S. Blott, A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces, Proceedings of the 24th International Conference on Very Large Data Bases, New York City, New York, 1998, pp. 194–205.

[13] V. Gaede, O. Guenther, Multidimensional access methods, ACM Computing Surv. (1998) 30(2) 170–231.

[14] D. Rafiei, A. Mendelzon, Efficient retrieval of similarity time sequences using dft, Proceedings of International Conference on Foundations of Data Organization, Kobe, Japan, 1998, pp. 249–257.