Ranked Subsequence Matching in Time-Series Databases

Wook-Shin Han
Department of Computer Engineering
Kyungpook National University
Republic of Korea
wshan@knu.ac.kr

Jinsoo Lee
Department of Computer Engineering
Kyungpook National University
Republic of Korea
jslee@www-db.knu.ac.kr

Yang-Sae Moon
Department of Computer Science
Kangwon National University
Republic of Korea
ysmoon@kangwon.ac.kr

Haifeng Jiang*
Google Inc.
Mountain View
California, USA
jianghf@google.com

ABSTRACT

Existing work on similar sequence matching has focused on either *whole* matching or *range* subsequence matching. In this paper, we present novel methods for *ranked sub*sequence matching under time warping, which finds top-k subsequences most similar to a query sequence from data sequences. To the best of our knowledge, this is the first and most sophisticated subsequence matching solution mentioned in the literature. Specifically, we first provide a new notion of the *minimum-distance matching-window pair* (*MDMWP*) and formally define the *mdmwp-distance*, a lower bound between a data subsequence and a query sequence. The mdmwp-distance can be computed prior to accessing the actual subsequence. Based on the mdmwp-distance, we then develop a ranked subsequence matching algorithm to prune unnecessary subsequence accesses. Next, to reduce random disk I/Os and bad buffer utilization, we develop a method of deferred group subsequence retrieval. We then derive another lower bound, the *window-group distance*, that can be used to effectively prune unnecessary subsequence accesses during deferred group-subsequence retrieval. Through extensive experiments with many data sets, we showcase the superiority of the proposed methods.

1. INTRODUCTION

Time-series data are of growing importance in many new database applications such as data mining and data warehousing [13, 20]. A time-series is a sequence of real numbers representing values at specific points in time. Typical examples of time-series data include music data, stock prices and network traffic data. The time-series data stored in a database are called data sequences.

*Work done while the author was with IBM Almaden Research Center

Various similarity models have been studied in similar sequence matching (finding data sequences similar to the given query sequence from the database). In this paper, we use the similarity model based on the dynamic time warping (DTW) distance [4, 24]. The DTW distance is one of the most robust and widely used measures for various applications such as query by humming [31], image searching [5], and speech recognition [21].

We develop a fast ranked subsequence matching solution for time-series databases using distances as the ranking method. Ranked subsequence matching finds top-k similar subsequences to a query sequence from data sequences. To the best of our knowledge, this is the first and foremost approach for such a matching, as all the existing methods have been developed only for either whole matching [1, 6, 11, 12, 15, 20, 31] or range subsequence matching [7, 16, 18, 19, 28, 29, 30]. Range subsequence matching finds all similar subsequences to a query sequence where their distances are less than or equal to the tolerance value ϵ. However, we argue that the range subsequence matching is very cumbersome to use in practice since it requires users to know the corresponding tolerance value in advance, in order to find top-k similar subsequences using the range subsequence matching. Furthermore, range subsequence matching first obtains candidate subsequences in the index level without accessing data sequences whereas ranked subsequence matching progressively accesses data sequences during index search. Thus, we need to develop 1) tight lower bounds for efficient matching, and 2) a method to avoid random disk I/Os. Table 1 summarizes representative research on similar sequence matching. As illustrated in Table 1, the k-NN, i.e., ranked subsequence matching problem has not been solved yet.

Table 1: Comparison of the proposed methods and related work.

Category	Range query	k-NN query
Whole matching	[1, 6, 11, 12, 15, 20, 31]	[6, 11, 12]
Subsequence matching	[7, 16, 17, 18, 19, 28]	$\times$

To perform ranked subsequence matching, we exploit the window construction method of our earlier work, DualMatch [18], which is both efficient and simple for range subsequence matching. That is, we divide data sequences into *disjoint windows* and query sequences into *sliding windows*[1]. We first propose a concept of the *minimum-distance matching-window pair* and derive a lower bound, called the *mdmwp-distance*, based on this concept. The minimum-distance matching-window pair for a query sequence and a subsequence is a matching window pair, where the distance between the two windows in the pair is the minimum among all matching window pairs. Thus, as soon as we obtain the minimum-distance matching-window pair for a subsequence, we can derive the mdmwp-distance for the subsequence. Then, we can aggressively prune unnecessary subsequence access requests in the index level. We also validate our algorithm by proving the lower-boundness of the mdmwp-distance.

We then propose a novel optimization technique called *deferred group subsequence retrieval* to avoid excessive random disk I/Os and bad buffer utilization. Deferred group subsequence retrieval 1) delays a set of subsequence retrieval requests, 2) groups the requests by their corresponding subsequences, and 3) enables batch retrieval. Since we have accumulated many requests, we can access subsequences in a sequential fashion. In addition, by exploiting many delayed matching windows, we derive another lower bound, called the *window-group distance*, that can be used together with deferred group subsequence retrieval. Based on the window-group distance, we then propose another ranked subsequence matching algorithm. We also validate this algorithm by proving the lower-bound property of the window-group distance.

We performed extensive experiments using various data sets: a mixed data set containing 33 data sets of the UCR time-series archive [14]; a random walk data set; a stock data set; and a music data set. The results show that our algorithms outperform competing algorithms considerably up to orders of magnitude. This advantage is particularly large when the buffer size is small.

The remainder of this paper is organized as follows. Section 2 reviews DTW and existing work related to range sequence matching and k-NN search in time-series databases. Section 3 presents the mdmwp-distance and the ranked subsequence matching algorithms based on the distance. Section 4 presents an optimization technique to boost the ranked subsequence matching algorithm as well as the window-group distance. Section 5 presents the results of performance evaluation. Section 6 summarizes and concludes our paper.

2. BACKGROUND

In Table 2, we summarize the notation to be used throughout the paper. In Section 2.1 we present a review of DTW along with existing lower bounds for whole matching, and in Section 2.2 we describe presents a review of related work for similar sequence matching.

2.1 Review of DTW

In this section, we describe three distances, DTW and its two lower bounds, LB_Keogh [12] and LB_PAA [31]. LB_Keogh and LB_PAA can be used at the sequence level and at the index level, respectively.

Given two sequences S and Q of the same length, the DTW distance is recursively defined as follows. Figure 1(a) shows how DTW performs elastic alignments between two sequences.

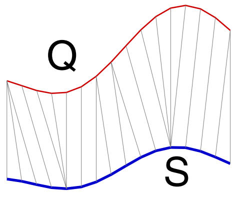
[1]Our method can be extended to exploit other subsequence matching methods [7, 19] by changing the window construction mechanism. (More on DualMatch will appear in Section 2.2.1.)

Table 2: Summary of notation.

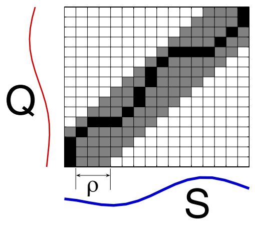
Symbols	Definitions
S_{sid}	A sequence whose identifier is *sid*
$Len(S)$	Length of sequence S
$S[i]$	The i-th entry of sequence S ($1 \le i \le Len(S)$)
$S[i:j]$	A subsequence of S, including entries from the i-th one to the j-th
ω	Length of the sliding/disjoint window
s_i	The i-th disjoint window of sequence S ($= S[(i-1)*\omega+1 : i*\omega], i \ge 1$)
$\langle\rangle$	empty sequence
$Rest(S)$	A subsequence of S, including entries from the second one to the last ($= S[2 : Len(S)]$)

$$DTW(\langle\rangle, \langle\rangle) = 0$$
$$DTW(S, \langle\rangle) = DTW(\langle\rangle, Q) = \infty$$
$$DTW(S,Q) = \sqrt[p]{|S[1]-Q[1]|^p + \min \begin{cases} DTW(Rest(S), Rest(Q)) \\ DTW(Rest(S), Q) \\ DTW(S, Rest(Q)) \end{cases}}$$

The DTW distance is computed by dynamic programming with a matrix as shown in Figure 1(b). The *warping path* is defined as a sequence of matrix elements, representing the optimal alignment for the two sequences. A matrix element (i, j) in the warping path represents an alignment between a query point $Q[i]$ and a data point $S[j]$. To avoid pathological alignments and to reduce the time complexity of DTW, global constraints such as the Sakoe-Chiba band constraint [24] and the Itakura Parallelogram constraint [10] are used to limit the scope of the warping path [12]. With the Sakoe-Chiba band constraint, the (i, j) matrix element becomes ∞ if $|i - j| > \rho$, where ρ is the *warping width*. DTW$_\rho$ denotes the DTW distance with a warping width ρ. We note that, if $\rho = 0$, the DTW distance becomes the L_p distance.



(a) DTW comparison.



(b) Matrix for DTW computation.

Figure 1: Illustration of DTW.

Before reviewing existing lower bounding techniques for whole matching, we define the notions of the query envelope [31] and piecewise aggregate approximation (PAA) [12, 30].

Definition 1. *The query envelope of a query Q,* $\mathbb{E}(Q)$*, with the warping width* ρ *consists of the upper and lower envelopes of Q and represents the region defined between the upper envelope U and the lower envelope L. The i-th element* $(L[i], U[i])$ *in* $\mathbb{E}(Q)$ *is defined as follows:*

$$L[i] = \min_{-\rho \le r \le \rho} (Q[i+r]), \quad U[i] = \max_{-\rho \le r \le \rho} (Q[i+r]) \quad \square$$

PAA converts the original sequence of length N into f ($f \ll N$) equal sized segments, and then stores the mean values of the segments. Formally, PAA of a data sequence S of length N, $\mathcal{P}(S)$, is represented as a time-series $[\overline{S[1]}, ..., \overline{S[f]}]$ of length f, where

$$\overline{S[i]} = \frac{f}{N} \sum_{j=\frac{N}{f}(i-1)+1}^{\frac{N}{f}i} S[j].$$

We can calculate the PAA of a query envelope by applying PAA to the original upper and lower query envelopes. The $i\,(1 \leq i \leq f)$-th element $(\overline{L[i]}, \overline{U[i]})$ in $\mathcal{P}(\mathbb{E}(Q))$ is given by:

$$\overline{L[i]} = \frac{f}{N} \sum_{j=\frac{N}{f}(i-1)+1}^{\frac{N}{f}i} L[j], \quad \overline{U[i]} = \frac{f}{N} \sum_{j=\frac{N}{f}(i-1)+1}^{\frac{N}{f}i} U[j].$$

Now we define the distance (called LB_Keogh) between a query envelope $\mathbb{E}(Q)$ and a data sequence S, which is the tightest lower bound for DTW that is used at the sequence level.

$$LB\_Keogh(\mathbb{E}(Q), S) = \sqrt[p]{\sum_{i=1}^{N} \begin{cases} |S[i] - U[i]|^p & \text{if } S[i] > U[i] \\ |S[i] - L[i]|^p & \text{if } S[i] < L[i] \\ 0 & \text{otherwise} \end{cases}}$$

As a lower bound that can be used at the index level, LB_PAA[2] is defined as the distance between the PAA of the query envelope $\mathcal{P}(\mathbb{E}(Q))$ and the PAA of the data sequence $\mathcal{P}(S)$.

$$LB\_PAA(\mathcal{P}(\mathbb{E}(Q)), \mathcal{P}(S)) = \sqrt[p]{\sum_{i=1}^{f} \frac{N}{f} \begin{cases} |\overline{S[i]} - \overline{U[i]}|^p & \text{if } \overline{S[i]} > \overline{U[i]} \\ |\overline{S[i]} - \overline{L[i]}|^p & \text{if } \overline{S[i]} < \overline{L[i]} \\ 0 & \text{otherwise} \end{cases}} \quad (1)$$

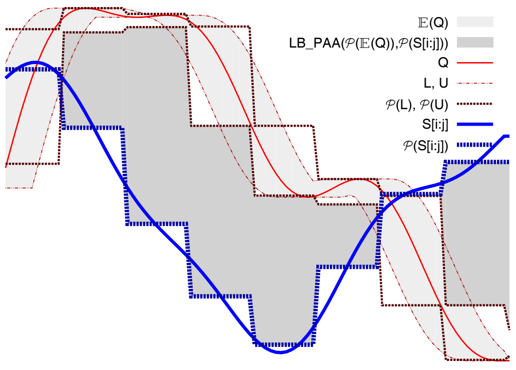


Figure 2: Illustration of LB_PAA.

To guarantee no false dismissal under DTW, we have the following lemma [31].

Lemma 1. *Given two sequences Q and S of the same length and a warping width ρ, the following equation holds:*

$$DTW_\rho(Q, S) \geq LB\_Keogh(\mathbb{E}(Q), S) \geq LB\_PAA(\mathcal{P}(\mathbb{E}(Q)), \mathcal{P}(S))$$

2.2 Related Work

2.2.1 Range Sequence Matching

We first review Agrawal et al.'s whole matching solution [1], from which many other similar sequence matching solutions including our algorithms have evolved. Their whole matching solution consists of index building and similar sequence matching processes. In the index building process, each data sequence is transformed into low-dimensional points using lower-dimensional transformation, and the transformed points are stored in an R*-tree [2]. The lower-dimensional transformation is used to avoid the high dimensionality problem [3, 27]. We can use DFT, DWT, PAA, or SVD as the lower-dimensional transformation.

In the similar sequence matching process, a query sequence is similarly transformed into a low-dimensional point, and a range query is constructed using the point and the given tolerance. Then, by evaluating the range query using the index, the candidates are identified. This method guarantees there be no false dismissal. It may, however, cause *false alarms* (i.e., candidates that do not qualify). Thus, for each candidate sequence obtained, the actual data sequence is accessed from the disk; the distance from the query sequence is computed; and the candidate is discarded whenever it is a false alarm. Our algorithms largely differ from this Agrawal et al.'s solution since they focused on whole matching with range queries, but we focused on subsequence matching with k-NN queries.

Faloutsos et al. [7] proposed a range subsequence matching solution (*FRM* in short) as a generalization of the whole matching solution. FRM uses the window construction method of dividing data sequences into sliding windows and a query sequence into disjoint windows. In the index building process, FRM divides data sequences into sliding windows, transforms each window into a low-dimensional point, and stores the point into the R*-tree. In the subsequence matching process, FRM divides a query sequence into disjoint windows; transforms each window into a low-dimensional point; makes a range query using that point and the given tolerance; and constructs a candidate set by searching the R*-tree. Finally, it identifies similar subsequences by eliminating false alarms. Wong and Wong [28] proposed a range subsequence matching under time warping by exploiting sliding windows. Their method has several problems: 1) their lower bound is loose in that only prefixes of all possible subsequences are indexed; 2) they do not exploit any dimension reduction technique, only suitable for very short queries due to dimensionality curse [3]; and 3) in their experiment, every query they tested has only one window (same as whole matching).

DualMatch [18, 16] and GeneralMatch [19], improve performance in range subsequence matching by using different window construction methods from FRM. Introducing the notion of *duality* in constructing windows, DualMatch, performs subsequence matching by dividing the data sequences into disjoint windows and the query sequence into sliding windows. GeneralMatch generalizes the concept of sliding windows and disjoint windows, and defines *J-sliding windows* and *J-disjoint windows* respectively, performing subsequence matching using these generalized windows. Except for a difference in the window construction mechanism, index building and subsequence matching algorithms of DualMatch and GeneralMatch are similar to those of FRM.

The algorithms to be proposed in this paper adapts the window construction mechanism of DualMatch, defined under the Euclidean distance metric, and extend it to work under time warping. We note that our ranked subsequence matching solutions differ from the previous subsequence matching solutions in the following two points: 1) we support ranked subsequence matching as opposed to range subsequence matching; and 2) we use the DTW distance as opposed to the Euclidean distance.

2.2.2 K-Nearest Neighbor Search

Hjaltason et al. [9] and Roussopoulos et al. [23] proposed the traditional k-NN search algorithms that use a minimum priority queue to find the k-nearest objects from a query object. In these algorithms, we first assume the object is stored in the multidimensional index as an MBR (minimum bounding rectangle). Each time, we then maintain the topmost k nodes/objects, which have the k-smallest distances from the query object, in the minimum priority

[2]We use Zhu and Shasha's LB_PAA [31], which is an optimized version of Keogh's LB_PAA [12].

queue. That is, the minimum priority queue maintains k records each of which consists of node/object O, its type, and the distance $D(Q, O)$ from the query object Q, and the records are sorted by the distance values. In the k-NN algorithms, we repeatedly replace the records of the queue with the new nearest ones, and eventually identify k objects that are nearest from the query object.

In the case of similar sequence matching, Keogh et al. [11] and Chan et al. [6] proposed k-NN search algorithms for the whole matching problem. Keogh et al. proposed a novel dimension reduction technique, called APCA (adaptive piecewise constant approximation), and they describe a k-NN whole matching algorithm based on the basic k-NN solutions [9, 23] in order to demonstrate superiority of their reduction technique. Chan et al. proposed another k-NN whole matching algorithm that first finds an upper bound of search range using Roussopoulos et al.'s k-NN solution, and then performs the range whole matching using the bound. These k-NN solutions differ from ours in that they handle the whole matching problem only, while our solutions focus on *subsequence matching* which is generalization of whole matching [19].

Keogh [12] proposed LB_Keogh and LB_PAA, tight lower bounds under time-warping, and a k-NN whole matching solution based on these bounds. Zhu and Shasha [31] proposed another LB_PAA, an optimal version of Keogh's LB_PAA, and a range whole matching solution based on LB_PAA. As we explained in Section 3, our solutions exploited these two lower bounds. However, our solutions differ from both solutions: both solutions are for whole matching, but ours for subsequence matching; and Zhu and Shasha's solution was for range whole matching, but ours for ranked subsequence matching.

We summarized in Table 1 (in Section 1) representative research works on similar sequence matching. As illustrated in Table 1, our proposed algorithms are the first approach that solves the k-NN, i.e., ranked, subsequence matching problem.

3. RANKED SUBSEQUENCE MATCHING

In this section, we propose ranked subsequence matching algorithms. In Section 3.1, we introduce two algorithms that extend earlier range subsequence matching and k-NN algorithms to ranked subsequence matching by exploiting the window construction mechanism in DualMatch. In Section 3.2, we provide notions of minimum-distance matching-window pair and mdmwp-distance, and propose a new ranked subsequence algorithm based on these notions.

Before we present ranked subsequence matching algorithms, we give a formal definition of our problem:

Problem Definition 1. *Given n data sequences S_1, ..., S_n of variable lengths, a query sequence Q, a number k, and a warping width ρ, find k-nearest data subsequences for Q by using DTW_ρ.*

Here, we do not allow Q of length l_1 to match with a subsequence S of length l_2 where $l_1 \neq l_2$. To match Q with subsequences of different lengths, we need to scale Q with reasonable scale factors (to prevent pathological scaling). While this is out of our scope, one can support such matching on top of our framework.

3.1 DualMatchTopK and RangeTopK

We propose two ranked subsequence matching algorithms that can be devised by applying DualMatch [18] to ranked whole matching algorithms [6, 12]. We first propose a ranked subsequence matching algorithm *DualMatchTopK* that applies the window construction mechanism of DualMatch to Keogh et al.'s ranked whole matching algorithm [12]. We then propose another algorithm *RangeTopK* that applies *DualMatchTopK* to Chan et al.'s ranked whole matching algorithm [6].

Like Keogh et al.'s ranked whole matching algorithm, *DualMatchTopK* uses the traditional k-NN solutions [9, 23, 25] for the ranked subsequence matching. (For the detailed explanation on traditional k-NN solutions, readers are referred to Section 2.2.2.) In Keogh et al.'s algorithm, they first construct a multidimensional index using lower-dimensional transformations of fixed-length data sequences, and subsequently evaluate the traditional k-NN query over the index for the given query sequence of the same length. In subsequence matching, however, data and query sequences can have arbitrary lengths. Thus, to adapt Keogh et al.'s whole matching algorithm to subsequence matching, we need to exploit the window construction method in subsequence matching. For the window construction method, we exploit that of DualMatch, which minimizes the index size. That is, we first divide data sequences into disjoint windows and store the windows in the multidimensional index through lower-dimensional transformations. Thereafter, to support DTW, we construct the query envelope for a query sequence, and then divide the query envelope into sliding windows, and find k-NN subsequences from the index using the traditional k-NN algorithm.

Algorithm 1 shows a ranked subsequence matching algorithm *DualMatchTopK*. In this algorithm, we use an in-memory minimum priority queue[3] whose entry is in the form of a quintuple *<obj, d, j, sid, off>*. Here, *obj* is the object type, i.e., one of a subsequence, a leaf entry, a leaf node, and a non-leaf node; *d* the distance between the object and the matching window in the subsequence; *j* the matching window number, i.e., the sliding window number; *sid* the corresponding sequence id; and *off* the offset of the corresponding subsequence. The inputs of the algorithm are a query sequence Q, the number k of subsequences to be returned, and a warping width ρ. According to the window construction method of DualMatch, we construct a query envelope $\mathbb{E}(Q)$ from Q and divide the query envelope $\mathbb{E}(Q)$ into sliding windows $\mathbb{E}(q_i)$. We then transform each sliding window $\mathbb{E}(q_i)$ into a low-dimensional window $\mathcal{P}(\mathbb{E}(q_i))$, and push the transformed window $\mathcal{P}(\mathbb{E}(q_i))$ into the priority queue (Lines 4-5). Next, we pop an entry from the queue one by one and perform the procedure of finding k-NN subsequences (Lines 6-23) as follows. 1) If *obj* is a subsequence, we append it to `result` and return the `result` if it already contains k-NN subsequences (Lines 8-11). 2) If *obj* is a leaf entry, we compute the offset of the corresponding subsequence and push the subsequence into the priority queue (Lines 12-15). Here, we call *SequenceRetrieval* function to first retrieve the subsequence from a data sequence, if it is not yet retrieved, and then push the candidate subsequence into the queue if both the LB_Keogh distance and the DTW distance are less then δ_{cur}, which is the DTW distance between Q and the top k-th subsequence obtained so far. 3) If *obj* is a leaf node, we push every leaf entry contained in the node into the queue (Lines 16-19). 4) If *obj* is a non-leaf node, we push every child entry of the node into the queue again (Lines 20-23).

In *DualMatchTopK*, pruning is executed just before an entry is pushed into the queue, i.e., just before Lines 19 and 23 of Algorithm 1 and Line 5 in *SequenceRetrieval* function. This is because entries whose lower bound distances are greater than δ_{cur} will not be included in the top-k subsequences. We note that δ_{cur} continually decreases as the top-k entries are changed. We also note

[3]We observe that the size of the priority queue can grow large. To tackle this problem, we propose priority queue minimization methods in which we group query points into several query MBRs (QMBRs) so that pairs of data points and QMBRs (rather than query points themselves) remain in the priority queue until they are popped. Due to space limit, we refer the reader to the extended version of our paper [8].

Algorithm 1 *DualMatchTopK*

```
Input: $Q$, $k$, $\rho$
Output: $k$-nearest data subsequences for $Q$
 1: Variable queue : Minimum priority queue
 2: Variable results : List
 3: Variable $\delta_{cur} \leftarrow \infty$; /*$\delta_{cur}$ is the DTW$_\rho$ distance between the $Q$
    and the top $k$-th subsequence obtained so far*/
 4: for each $i$-th sliding window $\mathbb{E}(q_i)$ in $\mathbb{E}(Q)$ do
 5:   queue.Push($\langle$RootNode,MINDIST($\mathcal{P}(\mathbb{E}(q_i))$,RootNode),$i$,$-1$,$-1\rangle$);
 6: while not queue.IsEmpty() do
 7:   $\langle$obj, d, j, sid, off$\rangle \leftarrow$ queue.Pop();
 8:   if obj is a subsequence then
 9:     add obj to results;
10:     if $|results| = k$ then
11:       return results;
12:   else if obj is a leaf entry then
13:     soff $\leftarrow$ off $- j + 1$; /*start offset*/
14:     eoff $\leftarrow$ soff $+ Len(Q) - 1$; /*end offset*/
15:     SequenceRetrieval(queue, soff, eoff, sid, $\delta_{cur}$, $\rho$);
16:   else if obj is a leaf node LN then
17:     for each leaf entry $E$ $\langle$Point $P$, SeqID sid2, offset off2$\rangle$ in LN do
18:       if LB_PAA($\mathcal{P}(\mathbb{E}(q_j))$,$P$) $< \delta_{cur}$ then
19:         queue.Push($\langle E$, LB_PAA($\mathcal{P}(\mathbb{E}(q_j))$,$P$), $j$, sid2, off2$\rangle$);
20:   else
21:     for each child node $E$ $\langle$MBR $M$, Child $ptr\rangle$ in obj do
22:       if MINDIST($\mathcal{P}(\mathbb{E}(q_j))$,$M$) $< \delta_{cur}$ then
23:         queue.Push($\langle E$, MINDIST($\mathcal{P}(\mathbb{E}(q_j))$,$M$), $j$, $-1$, $-1\rangle$);
```

Function 2 *SequenceRetrieval*

```
Input: queue, soff, eoff, sid, $\delta_{cur}$, $\rho$
1: if $Sub(= S_{sid}[soff : eoff])$ is not yet retrieved then
2:   retrieve $Sub$ from $S_{sid}[soff : eoff]$;
3:   if LB_Keogh($\mathbb{E}(Q)$,$Sub$) $< \delta_{cur}$ then
4:     if DTW$_\rho$($Q$,$Sub$) $< \delta_{cur}$ then
5:       queue.Push($Sub$, DTW$_\rho$($Q$,$Sub$),$-1$, sid,soff);
6:       update $\delta_{cur}$;
```

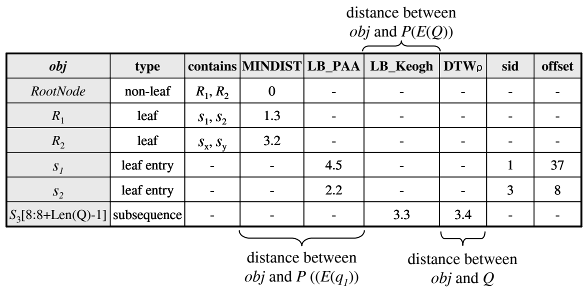
that MINDIST is a lower bounding distance function between a minimum bounding rectangle (MBR) M and an enveloped query window $\mathcal{P}(\mathbb{E}(q_j))$ [4].

Example 1. *Figure 3 depicts how DualMatchTopK operates using an example. Figure 3(a) shows the objects, object types and distances, and Figures 3(b)∼(d) show how the minimum priority queue changes according to object types of entries popped from the queue. Note that DualMatchTopK computes the distances using different distance functions (MINDIST, LB_PAA, LB_Keogh or DTW$_\rho$) according to object types of entries popped.*

- ***When a non-leaf node** RootNode **is popped** (Figure 3(b)): **1.** Pop $\langle$RootNode, 0, 1, -1, -1$\rangle$ from the queue (Line 7). **2.** Compute the MINDIST distance between each child node of RootNode and $\mathcal{P}(\mathbb{E}(q_i))$; push $\langle R_1$, 1.3, 1, -1, -1$\rangle$ and $\langle R_2$, 3.2, 1, -1, -1$\rangle$ into the queue. Here, we do not prune any entry since δ_{cur} is ∞ (Lines 21∼23).*

- ***When a leaf node** R_1 **is popped** (Figure 3(c)): **1.** Pop $\langle R_1$, d=1.3, j=1, -1, -1$\rangle$(Line 7). **2.** Compute the LB_PAA distance for each leaf entry contained in R_1 and $\mathcal{P}(\mathbb{E}(q_i))$; prune the leaf entry s_1 since its LB_PAA distance (=4.5) is greater than δ_{cur} (=4.0); but push $\langle s_2$, 2.2, 1, 3, 8$\rangle$ since its LB_PAA distance (=2.2) is less than δ_{cur} (Lines 17∼19).*

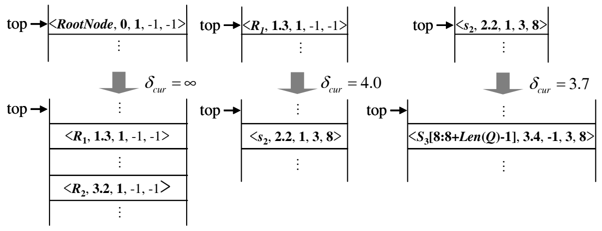
- ***When a leaf entry** s_2 **is popped** (Figure 3(d)): **1.** Pop $\langle s_2$, d=2.2, j=1, sid=3, off=8$\rangle$ from the queue (Line 7). **2.** Compute the start and end offsets of the matched subsequence and retrieve the corresponding subsequence; compute LB_Keogh distance between the subsequence and the query envelope $\mathbb{E}(Q)$ (we do not prune this subsequence since its LB_Keogh distance (=3.3) is less than δ_{cur} (=3.7)); compute DTW$_\rho$ distance (=3.4) between the subsequence and query sequence Q; and push $\langle S_3$[8:8+Len(Q)-1], 3.4, -1, 3, 8$\rangle$ and update δ_{cur} as 3.4 (Lines 12∼15).* □



distance between *obj* and *P*(*E*(*Q*))
distance between *obj* and *P* ((*E*(q_1))
distance between *obj* and *Q*

obj	type	contains	MINDIST	LB_PAA	LB_Keogh	DTW$_\rho$	sid	offset
RootNode	non-leaf	R_1, R_2	0	-	-	-	-	-
R_1	leaf	s_1, s_2	1.3	-	-	-	-	-
R_2	leaf	s_x, s_y	3.2	-	-	-	-	-
s_1	leaf entry	-	-	4.5	-	-	1	37
s_2	leaf entry	-	-	2.2	-	-	3	8
S_3[8:8+Len(Q)-1]	subsequence	-	-	-	3.3	3.4	-	-

(a) objects, object types, and several distances.



(b) when *RootNode* is popped. (c) when the leaf node R_1 is popped. (d) when the leaf entry s_2 is popped.

Figure 3: An example of ***DualMatchTopK*****.**

As an alternative algorithm, we propose *RangeTopK*, which obtains an upper bound ϵ first corresponding to the top-kth distance at the index level, and then finds top-k subsequences using the range subsequence matching algorithm with ϵ. Algorithm 3 shows *RangeTopK*, which consists of two steps. First, it finds k-nearest candidates in the index level (Lines 5-22). Second, it retrieves all k candidate subsequences; sets the distance between the k-th subsequence and the query sequence to the user-specified tolerance ϵ as an upper bound; and extracts the actual k-NN subsequences by evaluating the range subsequence matching algorithm (Lines 23-24).

3.2 Minimum Distance Matching Window Pair and Pruning Algorithm

We can optimize *DualMatchTopK* to prevent the retrieval of unnecessary subsequences. Unlike whole matching, in *DualMatchTopK* we divide a long sequence into smaller windows and use the windows rather than the sequence itself in constructing and searching the index [18]. Likewise, since MBRs or points stored in the index represent the windows rather than the sequences, we cannot prune the index search space before retrieving the actual subsequences. For this reason, many unnecessary subsequences are likely to be retrieved. To solve this problem, we propose an approach that prunes much of the search space while searching the index, consequently reducing the number of subsequences retrieved.

In order to prune the index search space, we present the novel notion of *minimum-distance matching-window pair*, *MDMWP* in short. MDMWP is the window pair whose distance is the minimum among all the window pairs when comparing two sequences by dividing windows. We formally define MDMWP as follows.

Algorithm 3 *RangeTopK*

```
Input: $Q$, $k$, $\rho$
Output: results ($k$-nearest data subsequences for $Q$)
 1: Variable queue : Minimum priority queue;
 2: Variable max_queue : Maximum priority queue;
 3: Variable candidates : List;
 4: Variable $\epsilon$;
 5: for each $i$-th sliding window $\mathbb{E}(q_i)$ in $\mathbb{E}(Q)$ do
 6:     queue.Push(⟨RootNode,MINDIST($\mathcal{P}(\mathbb{E}(q_i))$,RootNode),$i$,$-1$,$-1$⟩);
 7: while not queue.IsEmpty() do
 8:     ⟨obj, d, j, sid, off⟩ ← queue.Pop();
 9:     if obj is a leaf entry then
10:         soff ← off − j + 1; /*start offset*/
11:         eoff ← soff + Len(Q) − 1; /*end offset*/
12:         max_queue.Push($Sub$, DTW$_\rho$($Q$,$Sub$),$-1$, sid,soff);
13:         if max_queue.Size() = $k$ then
14:             ⟨obj2, d2, j2, sid2, off2⟩ ← max_queue.Top();
15:             $\epsilon$ ← d2;
16:             break;
17:     else if obj is a leaf node LN then
18:         for each leaf entry E ⟨Point $P$, SeqID sid2, offset off2⟩ in LN do
19:             queue.Push(⟨$E$, LB_PAA($\mathcal{P}(\mathbb{E}(q_j))$,$P$), $j$, sid2, off2⟩);
20:     else if obj is a non-leaf node then
21:         for each child node E ⟨MBR $M$, Child ptr⟩ in obj do
22:             queue.Push(⟨$E$, MINDIST($\mathcal{P}(\mathbb{E}(q_j))$,$M$), $j$, $-1$, $-1$⟩);
23: candidates ← RangeScan($\mathbb{E}(Q)$, $\epsilon$);
24: results ← Refinement(candidates, $k$);
```

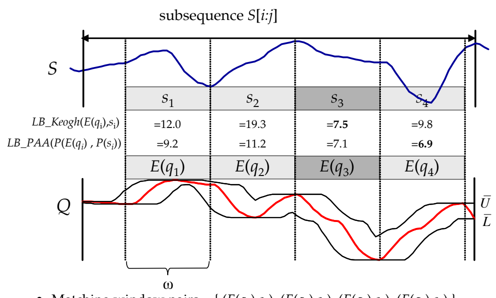
Definition 2. *Given a query envelope $\mathbb{E}(Q)$ and a data subsequence $S[i:j]$, a matching window pair is the pair of a disjoint window in $S[i:j]$ and the corresponding sliding window in $\mathbb{E}(Q)$. MDMWP is a matching window pair, where the distance between the two windows in the pair is the minimum among all matching window pairs.* □

The MDMWP can be defined not only in the original window space (i.e., in the high-dimensional space) but also in the transformed window space (i.e., in the low-dimensional space). We note that we use LB_Keogh and LB_PAA as distance functions in the original window space and in the transformed window space, respectively. Example 2 shows one MDMWP in the original window space and another in the transformed window space.

Example 2. *Figure 4 shows an example of MDMWP and matching window pairs. As shown in the figure, a query envelope $\mathbb{E}(Q)$ and the corresponding subsequence $S[i:j]$ have four matching window pairs, $(\mathbb{E}(q_1), s_1)$, $\cdots$, and $(\mathbb{E}(q_4), s_4)$. Here, the third one $(\mathbb{E}(q_3), s_3)$ is the MDMWP of $\mathbb{E}(Q)$ and S in the original window space since its distance $LB\_Keogh(\mathbb{E}(q_3), s_3)$ is the smallest among $(\mathbb{E}(q_i), s_i)$'s. However, the MDMWP of $\mathcal{P}(\mathbb{E}(Q))$ and $\mathcal{P}(S)$ in the transformed window space is the last one since the corresponding distance $LB\_PAA(\mathcal{P}(\mathbb{E}(q_4)), \mathcal{P}(s_4))$ is the smallest among $(\mathcal{P}(\mathbb{E}(q_i)), \mathcal{P}(s_i))$'s.* □

Using the concept of MDMWP, we can prune much of the index search space of *DualMatchTopK*. If we know MDMWP, we can then compute a lower bound of the distance between the query sequence and the corresponding subsequence in advance. To further illustrate this concept, we provide Definition 3 and Lemma 3.

Definition 3. *Given a query envelope $\mathbb{E}(Q)$ and a data subsequence $S[i:j]$, if MDMWP of $\mathcal{P}(\mathbb{E}(Q))$ and $\mathcal{P}(S[i:j])$ is $(\mathcal{P}(\mathbb{E}(q_m)), \mathcal{P}(s_m))$, then the mdmwp-distance of $\mathcal{P}(\mathbb{E}(Q))$ and $\mathcal{P}(S[i:j])$ is defined as $\sqrt[p]{r}\times$ $LB\_PAA(\mathcal{P}(\mathbb{E}(q_m)), \mathcal{P}(s_m))$ where $r = \lfloor (Len(Q)+1)/\omega \rfloor - 1$.* □



- Matching window pairs = { $(E(q_1),s_1)$, $(E(q_2),s_2)$, $(E(q_3),s_3)$, $(E(q_4),s_4)$ }
- MDMWP of $E(Q)$ and S = $(E(q_3),s_3)$
- MDMWP of $P(E(Q))$ and $P(S)$ = $(P(E(q_4)), P(s_4))$

Figure 4: An example of MDMWP.

Now, we need the following lemmas to guarantee the correctness of our approach, i.e., no false dismissals.

Lemma 2. *Given a query envelope $\mathbb{E}(Q)$ and a data subsequence $S[i:j]$, the following Eq. (2) holds;*

$$LB\_Keogh(\mathbb{E}(Q), S[i:j]) \geq LB\_PAA(\mathcal{P}(\mathbb{E}(q_1\cdots q_r)), \mathcal{P}(s_1\cdots s_r)) \tag{2}$$

where $r = \lfloor (Len(Q)+1)/\omega \rfloor - 1$, and $(\mathbb{E}(q_i), s_i)$ is the i-th matching window pair.

PROOF: By the window consruction method of *DualMatch*, the data subsequence $S[i:j]$ must include at least r disjoint windows s_1, ..., s_r, and also (possibly null) subsequences s_h(at the head) and s_t(at the tail). Thus, $S[i:j]$ can be represented as $s_h s_1 \cdots s_r s_t$. Similarly, $\mathbb{E}(Q)$ can be represented as $\mathbb{E}(q_h q_1 \cdots q_r q_t)$. Thus, we have

$$LB\_PAA(\mathcal{P}(\mathbb{E}(Q)), \mathcal{P}(S[i:j])) = LB\_PAA(\mathcal{P}(\mathbb{E}(q_h q_1 \cdots q_r q_t)), \mathcal{P}(s_h s_1 \cdots s_r s_t)) \tag{3}$$

Here, if we omit the two matching subsequences pairs $(\mathbb{E}(q_h), s_h)$ and $(\mathbb{E}(q_t), s_t)$ from Eq.(3), we have Eq.(4), since LB_PAA is a monotonic increasing function of the sequence length.

$$LB\_PAA(\mathcal{P}(\mathbb{E}(Q)), \mathcal{P}(S[i:j])) \geq LB\_PAA(\mathcal{P}(\mathbb{E}(q_1 \cdots q_r)), \mathcal{P}(s_1 \cdots s_r)) \tag{4}$$

By Lemma 1, Eq.(2) holds, and this completes the proof. □

Lemma 3. *Given a query envelope $\mathbb{E}(Q)$ and a data subsequence $S[i:j]$, the following Eq. (5) holds:*

$$DTW_\rho(Q, S[i:j]) \geq mdmwp\text{–}distance(\mathcal{P}(\mathbb{E}(Q)), \mathcal{P}(S[i:j])) \tag{5}$$

PROOF: We obtain the following (in)equations using Lemmas 1 and 2.

$$\begin{aligned} & DTW_\rho(Q, S[i:j]) \\ & \geq LB\_Keogh(\mathbb{E}(Q), S[i:j]) && \text{(by Lemma 1)} \\ & \geq LB\_PAA(\mathcal{P}(\mathbb{E}(q_1\cdots q_r)), \mathcal{P}(s_1\cdots s_r)) && \text{(by Lemma 2)} \end{aligned}$$

Since the length of $s_1 \cdots s_r$ is $r \times \omega$ and the length of $\mathcal{P}(s_1 \cdots s_r)$ is $r \times f$, we have Eq.(6). Here, ω is the length of a window; $\overline{L_k}$ and $\overline{U_k}$ are the lower and upper envelopes of $\mathcal{P}(\mathbb{E}(q_k))$, repectively; and $\overline{s_k}$ is $\mathcal{P}(s_k)$.

$$\begin{aligned} & LB\_PAA(\mathcal{P}(\mathbb{E}(q_1\cdots q_r)), \mathcal{P}(s_1\cdots s_r)) \\ & = \sqrt[p]{\sum_{k=1}^{r}\sum_{l=1}^{f}\frac{\omega}{f}\begin{cases} |\overline{s_k[l]} - \overline{U_k[l]}|^p & \text{if } \overline{s_k[l]} > \overline{U_k[l]} \\ |\overline{s_k[l]} - \overline{L_k[l]}|^p & \text{if } \overline{s_k[l]} < \overline{L_k[l]} \\ 0 \text{ otherwise} \end{cases}} \qquad (6) \\ & = \sqrt[p]{\sum_{k=1}^{r} LB\_PAA(\mathcal{P}(\mathbb{E}(q_k)), \mathcal{P}(s_k))^p} \qquad \text{(by Eq. (1))} \end{aligned}$$

Here, let MDMWP of $(\mathcal{P}(\mathbb{E}(Q)), \mathcal{P}(S[i:j]))$ be $(\mathcal{P}(\mathbb{E}(q_m)), \mathcal{P}(s_m))$, then $LB\_PAA(\mathcal{P}(\mathbb{E}(q_m)), \mathcal{P}(s_m))$ is less than or equal to LB_PAA $(\mathcal{P}(\mathbb{E}(q_k)), \mathcal{P}(s_k))$ for all k $(\leq r)$.

$$\begin{aligned}
&LB\_PAA(\mathcal{P}(\mathbb{E}(q_1 \cdots q_r)), \mathcal{P}(s_1 \cdots s_r)) \\
&\geq \sqrt[p]{\sum_{k=1}^{r} LB\_PAA(\mathcal{P}(\mathbb{E}(q_m)), \mathcal{P}(s_m))^p} \\
&= \sqrt[p]{r \cdot LB\_PAA(\mathcal{P}(\mathbb{E}(q_m)), \mathcal{P}(s_m))^p} \\
&= mdmwp\text{--}distance(\mathcal{P}(\mathbb{E}(Q)), \mathcal{P}(S[i:j])) \qquad (7)
\end{aligned}$$

Thus, Eq. (7) holds, and this completes the proof. □

To prune the unnecessary index search space, we use Theorem 1, which applies Lemma 3 to *DualMatchTopK*.

Theorem 1. *Suppose that the current popped entry is given by <obj, d, j, sid, off> in DualMatchTopK, where obj is a leaf entry and the corresponding subsequence for obj is not yet retrieved. Let δ_{cur} be the DTW$_\rho$ distance between the query sequence and the top k-th data subsequence obtained so far. If $\sqrt[p]{r}$ $\times$ $LB\_LAA(\mathcal{P}(\mathbb{E}(q_j)), \mathcal{P}(S_{sid}[off : off + \omega - 1]))$ is greater than δ_{cur}, then the corresponding subsequence is not included in the top-k subsequences. Here, q_j is the j-th sliding window of Q, and $r = \lfloor (Len(Q)+1)/\omega \rfloor - 1$.*

PROOF: We first show by contradiction that the pair $(\mathcal{P}(\mathbb{E}(q_j)),$ $\mathcal{P}(S_{sid}[off : off + \omega - 1]))$ must be MDMWP. Assume that this pair is not MDMWP and that the subsequence for this pair has not already been retrieved. However, since the distances of all popped entries are ordered by their distance values, MDMWP for the subsequence must have been popped earlier than this pair, and thus, the subsequence has already been retrieved. This contradicts the assumption above. Therefore, the pair must be MDMWP. We then prove the theorem using Lemma 3. According to Lemma 3, since $(\mathcal{P}(\mathbb{E}(q_j)), \mathcal{P}(S_{sid}[off : off + \omega - 1]))$ is MDMWP, the DTW$_\rho$ distance between the corresponding subsequence and the query sequence, $DTW_\rho(Q, S_{sid}[off - j + 1 : off - j + Len(Q)])$, is greater than or equal to the mdmwp-distance $\sqrt[p]{r} \times$ LB_PAA $(\mathcal{P}(\mathbb{E}(q_j)), \mathcal{P}(S[off : off + \omega - 1]))$. Thus, if the mdmwp-distance is greater than δ_{cur}, the DTW$_\rho$ distance between the corresponding subsequence and the query sequence is also greater than δ_{cur}, and accordingly, that subsequence cannot be included in the `results` of k-NN subsequences. □

We present the *AdvTopK* algorithm, which incorporates the mdmwp-distance based pruning technique in *DualMatchTopK*. We only need one modification in *DualMatchTopK*. That is, we replace *SequenceRetrieval* with *MDMWP-Retrieval*. Function *MDMWP-Retrieval* includes an additional pruning step in Line 3. We can prune the subsequence without retrieval if its mdmwp-distance is greater than δ_{cur} (Line 3). We note that we still have to use a loose lower bound not multiplying by $\sqrt[p]{r}$ in Lines 18 and 22 in Algorithm 1 since we cannot guarantee that a given MBR contains entries belonging to MDMWPs.

Theorem 2. *Algorithm $AdvTopK$ correctly retrieves top-k subsequences in order.*

PROOF: *AdvTopK* is devised by applying MDMWP-based pruning to *DualMatchTopK*. Note that we maintain a priority queue to keep top-k subsequences in order. Thus, it is sufficient to show that Theorem 1 is correctly applied to MDMWP-Retrieval of *AdvTopK*. We add lines 2-3 in MDMWP-Retrieval for MDMWP-based pruning according to Theorem 1. Let the current popped entry be *<obj, d, j, sid, off>* in *AdvTopK*. To guarantee MDMWP-based pruning by Theorem 1, we must satisfy the following conditions: c$_1$) obj is of a leaf entry; c$_2$) the corresponding subsequence for obj is not yet retrieved. To satisfy c$_1$), MDMWP-Retrieval is called only when obj is of a leaf entry. To satisfy c$_2$), we have line 1 in *MDMWP-Retrieval*. Additionally, to perform MDMWP-distance computations only for MDMWPs, we have lines 9-10 in *MDMWP-Retrieval*. That is, if an MDMWP is pruned in line 3, we mark the corresponding subsequence as retrieved. Therefore, *AdvTopK* correctly retrieves top-k subsequences in order. □

Function 4 *MDMWP-Retrieval*

```
Input: queue, d, j, soff, eoff, sid, $\delta_{cur}$, $\rho$
1: if $Sub (= S_{sid}[soff : eoff])$ is not yet retrieved then
2:     mdmwp-dist ← $\sqrt[p]{r}$ × d; /*$d = LB\_PAA(\mathcal{P}(\mathbb{E}(q_j)), \mathcal{P}(s_j))$*/
3:     if mdmwp-dist ≤ $\delta_{cur}$ then
4:         retrieve $Sub$ from $S_{sid}[soff : eoff]$;
5:         if LB_Keogh($\mathbb{E}(Q)$,$Sub$) < $\delta_{cur}$ then
6:             if DTW$_\rho$($Q$,$Sub$) < $\delta_{cur}$ then
7:                 queue.Push($Sub$, DTW$_\rho$($Q$,$Sub$), −1, sid, soff);
8:                 update $\delta_{cur}$;
9:     else
10:        mark $S_{sid}[soff : eoff]$ as retrieved;
```

4. DEFERRED GROUP SUBSEQUENCE RETRIEVAL

4.1 Concept

Although we prune a considerable number of unnecessary subsequence accesses at the index level, we still encounter two performance problems: 1) excessive random disk I/Os and 2) bad buffer utilization. When we retrieve subsequences one by one to refine candidates, accesses to the data pages for the candidates are likely to be unclustered, making each access incur a random disk I/O. More seriously, if we have a limited buffer size (especially, in a muti-user environment), these random access I/Os would result in low locality, and thus, we might have to read same data pages repeatedly from disk. Suppose that we have two buffer pages and six page accesses requested: p_{30}, p_{10}, p_{20}, p_{30}, p_{10}, p_{20}. In such a case, sequential flooding [22] is bound to happen; all the pages have to be read from disk since the pages already fetched (i.e., p_{30} and p_{10}) are replaced before being reused.

To tackle this problem, we propose *deferred group subsequence retrieval* that delays a fixed size set of subsequence retrieval requests and enables batch retrieval. Since we have many requests delayed, we can access our requests in a sequential fashion. However, since we change the original subsequence access order, δ_{cur} is not reduced as fast as when we access the subsequences in the original order. On the other hand, by exploiting many delayed matching windows we can further tighten the lower bound for each subsequence.

For deferred group subsequence retrieval, we introduce a data structure called the *group subsequence access list* for storing all requests delayed for the next bulk access. Group subsequence access list is a list of window request groups. Each window request group G_a corresponds to one subsequence retrieval and consists of up to r $(= \lfloor (Len(Q)+1)/\omega \rfloor - 1)$ window access requests $s_{a,b}$, where $s_{a,b}$ represents a window access request for the b-th matching window of G_a. That is, all requests delayed are grouped by their starting offsets. Here, a hash-based technique is used for fast grouping.

Example 3. *Figure 5 shows a group subsequence access list. As depicted, all window accesses requested are grouped by their starting addresses. The second group G_2 has the three window access requests $(s_{2,1}, s_{2,3}, s_{2,4})$ and therefore waits for the second disjoint window request $(s_{2,2})$.* □

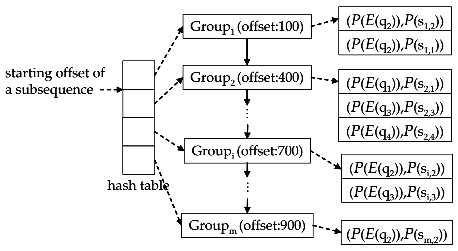


Figure 5: Example of a group subsequence access list.

In Definition 4, we formally define a new distance called the *window-group distance* that can be used to effectively prune unnecessary subsequence accesses during deferred group-subsequence retrieval. The window-group distance is derived by exploiting both delayed matching windows in each group and the largest distance in the group subsequence access list. Using the window-group distance, we can derive another lower bound in Theorem 3.

Definition 4. *Suppose that a group G_a in the group subsequence access list has m ($\le r$) windows W_m and that the last inserted entry in the list is $<\mathcal{P}(\mathbb{E}(q_c)), \mathcal{P}(s_c)>$. For each group corresponding to a subsequence $S[i:j]$ in the group subsequence access list, the window-group dist (WG-dist in short) between the query envelope $\mathbb{E}(Q)$ and the subsequence $S[i:j]$ is defined as*

$$\sqrt[p]{\begin{array}{l}\sum_{s_b \in W_m} LB\_PAA(\mathcal{P}(\mathbb{E}(q_b)), \mathcal{P}(s_b))^p \\ + LB\_PAA(\mathcal{P}(\mathbb{E}(q_c)), \mathcal{P}(s_c))^p \times (r-m)\end{array}}$$

where $r = \lfloor (Len(Q)+1)/\omega \rfloor - 1$. □

Theorem 3. *Given a query envelope $\mathbb{E}(Q)$ and a data subsequence $S[i:j]$, then the following Eq. (8) holds:*

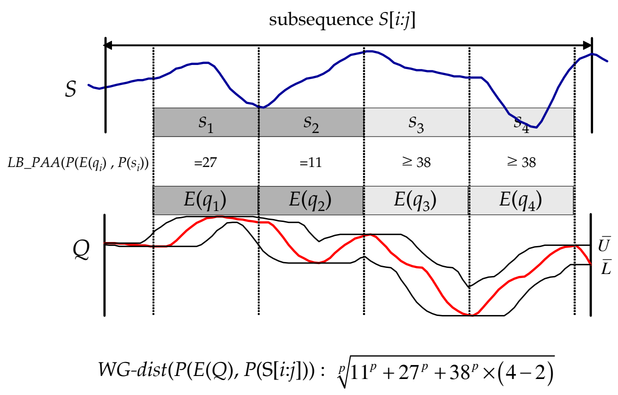
$$DTW_\rho(Q, S[i:j]) \ge WG\text{-}dist(\mathcal{P}(\mathbb{E}(Q)), \mathcal{P}(S[i:j])) \qquad (8)$$

PROOF: Here, the group G_a waits for $r-m$ window requests for this group. Since all window access requests popped from the priority queue are ordered by distance, the LB_PAA distance for any remaining window request is larger than the LB_PAA distance between $\mathcal{P}(s_c)$ and $\mathcal{P}(\mathbb{E}(q_c))$. Thus, the equation preserves a lower bound of the distance between the query and the subsequence corresponding to the group G_a. □

Example 4. *Recall G_1 in Figure 5. In Figure 6, suppose that the distances for the two requests are 11 and 27, respectively, and that $s_{2,4}$ in G_2 is the last inserted window request in the group subsequence access list, and its distance is 38. According to Theorem 3, the distances for the remaining window requests must be greater than or equal to 38. Thus, WG-dist for G_2 ($= \sqrt[p]{11^p + 27^p + 38^p \times (4-2)}$) can be derived as shown in Figure 6.* □

4.2 Putting It Altogether

We present the *DeferredTopK* algorithm, which incorporates the deferred group subsequence retrieval technique in *AdvTopK*. We need to modify two things in *AdvTopK*. 1) We replace *MDMWP-Retrieval* with *WGRetrieval*. 2) At the terminating condition in



subsequence $S[i:j]$
S
s_1
s_2
s_3
s_4
$LB\_PAA(P(E(q_i)), P(s_i))$
=27
=11
≥ 38
≥ 38
$E(q_1)$
$E(q_2)$
$E(q_3)$
$E(q_4)$
Q
$\bar{U}$
$\bar{L}$
$WG\text{-}dist(P(E(Q)), P(S[i:j]))$: $\sqrt[p]{11^p + 27^p + 38^p \times (4-2)}$

Figure 6: An example of *WG-dist* using Theorem 3.

Algorithm 1 (Line 10), if the size of a group sequence access list (*GSAL* in short) is greater than zero, we retrieve data subsequences $\texttt{result}_{\texttt{GSAL}}$ corresponding to remaining entries in *GSAL*, and return top-k subsequences among `result` and $\texttt{result}_{\texttt{GSAL}}$.

Function 5 shows the function *WGRetrieval* that uses a group subsequence access list *GSAL*. It adds a subsequence request to the corresponding group in the list until the size of *GSAL* reaches the user-defined maximum size (=MAX_GSAL_SIZE) of the list (Lines 1-3). Then, for each group corresponding to a subsequence, we calculate its lower bound (Lines 4-10). Here, we prune any subsequence request if its lower bound is greater than δ_{cur} (Line 11). Otherwise, we retrieve the subsequence in the data sequence, and push the result to the priority queue if both its LB_Keogh and DTW$_\rho$ distances are less than δ_{cur} (Lines 12-15).

Theorem 4. *DeferredTopK correctly retrieves top-k subsequences in order.*

PROOF SKETCH: Similarly to Theorem 2, i.e., by using the lower bounds and *DualMatchTopK*, we can prove correctness of *DeferredTopK*. We omit the detailed formal proof due to space limitation. □

Function 5 WG*Retrieval*

```
Input: queue, $d$, $soff$, $eoff$, $sid$, $\delta_{cur}$, $\rho$
 1: Static Variable GSAL : Group subsequence access list
 2: if $Sub(= S_{sid}[soff : eoff])$ is not yet retrieved then
 3:   GSAL[$\langle sid, soff \rangle$].Add(d);
 4:   if |GSAL| = MAX_GSAL_SIZE then
 5:     for each entry $\langle\langle sid2, soff2\rangle, winreqlist\rangle$ do
 6:       $eoff2 \leftarrow soff2 + Len(Q) - 1$;
 7:       $tmpDist \leftarrow (r - |winreqlist|) \times d^p$;
 8:       for each $win\_dist$ in $winreqlist$ do
 9:         $tmpDist \leftarrow tmpDist + win\_dist^p$;
10:       $wg\text{-}dist \leftarrow \sqrt[p]{tmpDist}$ ;
11:       if $wg\text{-}dist \le \delta_{cur}$ then
12:         retrieve $Sub2$ from $S_{sid2}[soff2 : eoff2]$;
13:         if LB_Keogh($\mathbb{E}(Q)$,$Sub2$) < $\delta_{cur}$ then
14:           if DTW$_\rho$($Q$,$Sub2$) < $\delta_{cur}$ then
15:             queue.Push($Sub2$, DTW$_\rho$($Q$,$Sub2$),−1, $sid2$, $soff2$);
16:             update $\delta_{cur}$;
17:       else
18:         mark $S_{sid2}[soff2 : eoff2]$ as retrieved;
19:     GSAL.Clear();
```

5. PERFORMANCE EVALUATION

We evaluate the performance of a sequential scan algorithm *SeqTopK* exploiting LB_Keogh and our four algorithms *DualMatchTopK*, *RangeTopK*, *AdvTopK* and *DeferredTopK*. We allocate memory of only 0.5% of the database size for the group subsequence access list in *DeferredTopK*. Our main objective in the experiments is to see performance gains for various parameters such as the buffer size and k. Section 5.1 describes the setup for the experiments, and Section 5.2 presents analysis of the experimental results.

We do not report the quality of our experimental results since our algorithms correctly find top-k subsequences in a ranked order in terms of DTW distances under our model (i.e., 100% quality guaranteed). However, in some applications the quality might be different from DTW distances, e.g., some poorly hummed melodies would match with unwanted melodies in databases. This issue is interesting, but is application-dependent.

5.1 Experiment Setup

As a main experimental data set, we use 33 data sets of different characteristics in the UCR time-series archive [14] used in [12, 31]. To evaluate the performance in subsequence matching, we construct a long sequence of 1,055,525 entries by concatenating all 33 UCR data sets. We call this UCR data set *UCR-DATA*. We have the following reasons to concatenate the datasets: 1) in order to analyze and compare with results in other datasets, the UCR dataset should be similar in size to the other datasets; 2) to show the effectiveness of our method regardless of heterogeneity of data and queries. In addition, we use three more data sets: one synthetic and two real data sets. The synthetic data set, used in FRM [7], DualMatch [18], and GeneralMatch [19], contains random walk data consisting of one million entries: 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 first real data set is a stock data set, also used in [7, 18, 19], consisting of 329,112 entries. We call this data set *STOCK-DATA*. The second real data set is a music data set consisting of 2,373,120 entries. We construct this music data set by extracting pitch data from 500 MIDI files. We call this data set *MUSIC-DATA*.

We use the number of candidates, the number of page accesses, and the wall clock time as the performance metrics. Table 3 summarizes experimental parameters and their values. We generate query sequences from the data sequence by taking subsequences of length $Len(Q)$ starting from random offsets as in [7, 18, 19]. We select the query sequences that have more than 90% filtering efficiency in searching the index, since the query sequences with lower filtering efficiency (i.e., less than 90%) are known to be inadequate to use the index [12]. However, *DeferredTopK* is as competitive as *SeqTopK* even when filtering efficiency is low. For this purpose, we perform another experiment for each of 33 queries having different filtering efficiency for UCR-DATA (see the results of Experiment 5). We omit results for this experiment due to space limitation. The reader may refer to [8].

All the experiments are done on Linux Kernel 2.6 PC with 512 Mbytes RAM and Pentium IV 2.8 GHz CPU. We use LRU as the buffer page replacement algorithm, and set the page size to 4096 bytes. To avoid the buffering effect of the OS file system and to guarantee actual disk I/Os, we use the O_DIRECT flag [26] when we open data and index files. As the multidimensional index, we use the R*-tree for all the index-based algorithms *DualMatchTopK*, *RangeTopK*, *AdvTopK*, and *DeferredTopK*. As we explained in Section 2, we use PAA as the feature extraction function and extract eight features from each window. For each query sequence, we set the warping width to 5% of length. We also performed experiments for varying the warping width, but the performance trends are similar, and thus we omit the results for brevity.

Table 3: Experimental parameters and their values.

Parameter	Default	Range
k	25	$5 \sim 50$
Buffer size	5%	$1\% \sim 10\%$
Len(Q)	384	256, 384, 512
ω (window size)	64	32, 64, 128

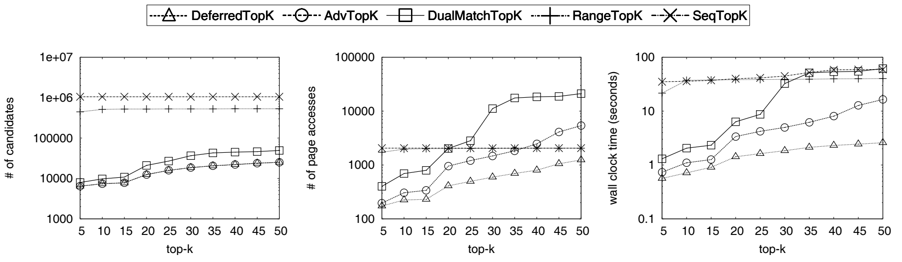
5.2 Experiments and Results

Experiment 1) (effect of k) Figure 7 shows the experimental results for UCR-DATA by varying k. Figure 7(a) shows the number of candidates; Figure 7(b) the number of page accesses; and Figure 7(c) the wall clock time.

As shown in Figure 7(a), *AdvTopK* and *DeferredTopK* significantly reduce the number of candidates by up to 163.8 times compared with *SeqTopK* and by up to 69.8 times compared with *RangeTopK*. *AdvTopK* and *DeferredTopK* also reduce the number of candidates by a factor of 2.1 compared with *DualMatchTopK*. This indicates that LB_PAA and our two lower bounds (mdmwp-distance and window-group distance) are very effective in pruning unnecessary subsequence access requests. *SeqTopK* shows the highest and most constant number of candidates, since it always scans the entire database. *RangeTopK* also shows a relatively large number of candidates. This means that the candidate set obtained in the index level is not tight enough to identify real top k similar subsequences.

In terms of the number of page accesses, *DeferredTopK* provides significant reduction by up to 25.2 times, 11.9 times, 10 times, and 4.3 times compared with *DualMatchTopK*, *SeqTopK*, *RangeTopK*, and *AdvTopK*, respectively. This shows that the deferred group subsequence retrieval is very effective in reducing the number of page accesses. As shown in Figure 7(b), *SeqTopK* shows a constant number of page accesses, since it sequentially retrieves all the data subsequences. *RangeTopK* also shows a constant number of page accesses, since it obtains a quite large number of candidates in the index level and retrieves the corresponding data subsequences in a sequential manner. *DualMatchTopK* and *AdvTopK*, however, retrieve the data subsequences that are not pruned in the index level, and thus, their numbers of page accesses increase as k increases, i.e., as the number of candidates increases. Due to this random access, the numbers of page accesses of *DualMatchTopK* and *AdvTopK* become larger than that of *SeqTopK* if k is greater than 25 and 40, respectively. In contrast, using the group subsequence access list, *DeferredTopK* retrieves the candidate subsequences as groups, that is, it accesses data pages in a sequential manner. This group access mechanism enables *DeferredTopK* to reduce the number of page accesses compared with *DualMatchTopK* and *AdvTopK*.

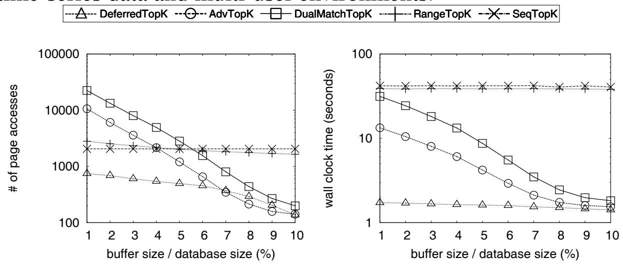
As shown in Figure 7(c), *DeferredTopK* significantly reduces the wall clock time compared with the other four algorithms (by up to 62.4 times, 52.3 times, 23.8 times, and 6.3 times compared with *SeqTopK*, *RangeTopK*, *DualMatchTopK*, and *AdvTopK*, respectively). In subsequence matching, the wall clock time is determined by two major factors: 1) the time for accessing disk pages and 2) the time for computing distances between query and candidate sequences. This means that the wall clock time is reflected by both the number of candidates and the number of page accesses. As shown in Figures 7(a) and 7(b), since *DeferredTopK* shows the best result in terms of both candidates and page accesses, it obviously outperforms the other algorithms in the wall clock time. Comparing Figure 7(c) with Figure 7(b), the increasing trend of *AdvTopK* and *DualMatchTopK* in the wall clock time is slightly slower than that in the number of page accesses. This is because these two algorithms significantly reduce the distance computation time due to a small number of candidates.



(a) Number of candidates. (b) Number of page accesses. (c) Wall clock time.

Figure 7: Experimental results for UCR-DATA by varying k.

Experiment 2) (effect of the buffer size) Figure 8 shows the experimental results for UCR-DATA by varying the buffer size. Figures 8(a) and 8(b) show the number of page accesses, and the wall clock time, respectively. We note that the number of candidates does not change for any buffer size, and thus, we omit the graph for the number of candidates. We note that the small buffer size is very crucial for both large (hundreds of gigabytes to terabytes) time-series data and multi-user environments.



(a) Number of page accesses. (b) Wall clock time.

Figure 8: Experimental results for UCR-DATA by varying the buffer size.

As the buffer size increases, the number of page accesses decreases for all the index-based algorithms *DualMatchTopK*, *RangeTopK*, *AdvTopK*, and *DeferredTopK*. This is obvious since, as the buffer size increases, the buffer hit-ratio in accessing candidate subsequences increases. We note that the *DeferredTopK* shows much better performance with a very small buffer size. This is the most desirable characteristic of the deferred group subsequence retrieval. That is, many random subsequence requests are accumulated in the group subsequence access list, being rescheduled to a sequential scan. The trend for the wall clock time is similar to that of the number of page accesses.

When the buffer size is small, *DualMatchTopK* is much worse than the other algorithms including *SeqTopK* in terms of page accesses. For example, when the buffer size is less than 6% in Figure 8(a), *DualMatchTopK* shows the worst result in page accesses, since it incurs the largest number of random accesses. However, due to reduction in the distance computation time (i.e., the number of candidates), in Figure 8(b), *DualMatchTopK* is better than *SeqTopK* and *RangeTopK*. On the other hand, *DeferredTopK* shows the best result both in the wall clock time and in page accesses, since it exploits both the pruning effect by lower bounds and the buffering effect by the group sequential access list. Specifically, *DeferredTopK* reduces the wall clock time by up to 28.6 times, 26.9 times, 18.2 times, and 7.7 times compared with *SeqTopK*, *RangeTopK*, *DualMatchTopK*, and *AdvTopK*, respectively.

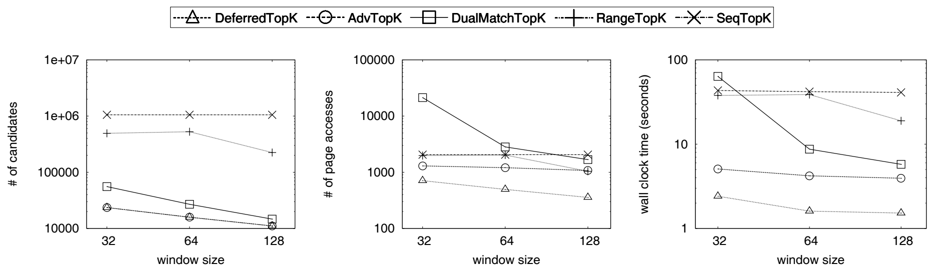
Experiment 3) (effect of the window size) Figure 9 shows the experimental results for UCR-DATA by varying the window size. Like Experiments 1 and 2, *SeqTopK* shows constant values in all three measures, since it fully scans the entire database and considers all possible subsequences, regardless of the window size. On the other hand, the results of the other four algorithms are changed according to the window size, since they use window construction mechanism [7, 18, 19] of subsequence matching. We note that, as the window size increases, all three measures of these four algorithms decrease. This decreasing trend is well explained by the *window size effect* [18]. That is, longer windows decrease the number of candidates, and accordingly, reducing the number of page accesses and the wall clock time.

As shown in Figure 9(c), *DeferredTopK* still outperforms the other four algorithms, regardless of the window size. It improves matching performance by up to 27.0 times, 26.4 times, 24.1 times, and 2.6 times compared with *SeqTopK*, *DualMatchTopK*, *RangeTopK*, and *AdvTopK*, respectively.

Experiment 4) (effect of the query length) Figure 10 shows the experimental results for UCR-DATA by varying the query length. In the case of *SeqTopK*, the number of candidates and the number of page accesses do not change according to the query length (actually, the number of candidates is only slightly changed, since the number of all possible subsequences is given by $Len(S) - Len(Q) + 1$). The wall clock time of *SeqTopK*, however, increases according to the query length, since the longer length requires more operations in computing the DTW distance. *RangeTopK* shows a similar trend to *SeqTopK*, since, as we explained in Experiment 1, it has sequential characteristics in accessing candidate subsequences.

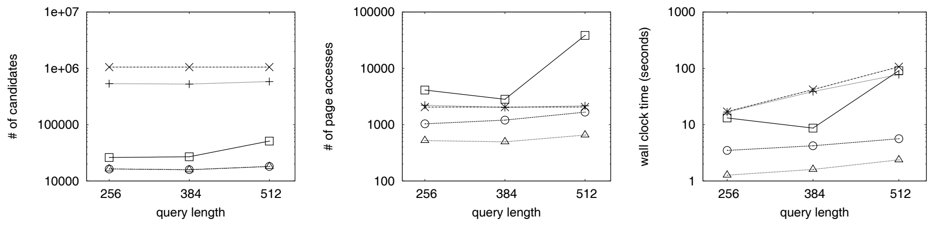
In cases of *DualMatchTopK*, *AdvTopK*, and *DeferredTopK*, the number of candidates slightly increases according to the query length. This increasing trend is also explained by the window size effect. That is, as the query length increases, the relative size of the corresponding window decreases, and thus, the more candidates occur due to the window size effect. Due to the increase in the number of candidates, the number of page accesses and the wall clock time also increase for the larger query length. We note that, in Figures 10(b) and 10(c), the increasing slope of *DualMatchTopK* is the sharpest compared with *AdvTopK* and *DeferredTopK*, since it has the least efficient buffer utilization. Like the previous experimental results, we can see that *DeferredTopK* improves matching performance by one or two orders of magnitude compared with the other algorithms.

Experiment 5) (effect of different data sets) Figures 11 and 12 show the experimental results for WALK-DATA and MUSIC-DATA, respectively. We omit the result for STOCK-DATA since the trend in STOCK-DATA is similar to the one in WALK-DATA. We use the same parameter values as in Experiment 1. Because the results in the figures show such a similar tendency to those of UCR-DATA in Figure 7, we omit the detailed description of the performance results.



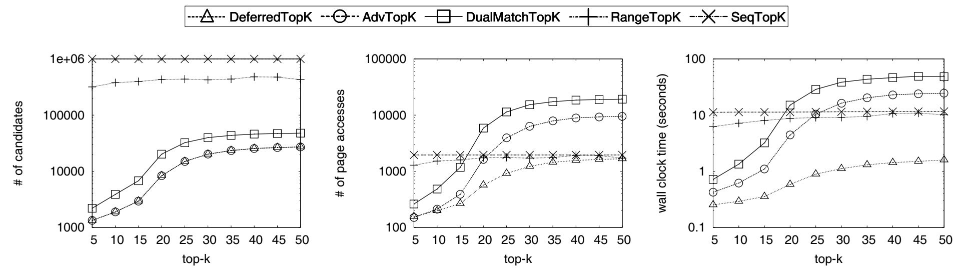
(a) Number of candidates. (b) Number of page accesses. (c) Wall clock time.

Figure 9: Experimental results for UCR-DATA by varying the window size.



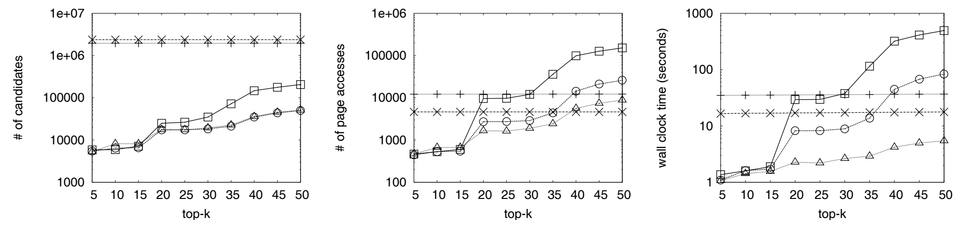
(a) Number of candidates. (b) Number of page accesses. (c) Wall clock time.

Figure 10: Experimental results for UCR-DATA by varying the query length.



(a) Number of candidates. (b) Number of page accesses. (c) Wall clock time.

Figure 11: Experimental results for WALK-DATA by varying k.



(a) Number of candidates. (b) Number of page accesses. (c) Wall clock time.

Figure 12: Experimental results for MUSIC-DATA by varying k.

6. CONCLUSIONS

In this paper, we presented novel ranked subsequence matching methods for time-series databases. We showed that our advanced algorithms reduce false alarms and improve performance significantly compared with the competing algorithms.

We first proposed two ranked subsequence matching algorithms *DualMatchTopK* and *RangeTopK* by adapting the window construction mechanism of DualMatch defined under Euclidean distance metric to work under time warping, and applying it to the previous k-NN whole matching solutions. To improve the pruning power in searching the index, we then proposed a novel notion of the minimum-distance matching-window pair and derived a lower bound *mdmwp-distance* based on the notion in Theorem 1. Using the mdmwp-distance, we proposed an advanced algorithm *AdvTopK* and proved its correctness in Theorem 2. After then, we proposed the deferred group subsequence retrieval to avoid excessive random disk I/Os and bad buffer utilization. Deferred group subsequence retrieval delayed a set of subsequence retrieval requests, grouped the requests by their corresponding subsequences, and enabled batch retrieval. Since we accumulated many requests, we could access subsequences in a sequential access fashion. In addition, by exploiting many delayed matching windows, we derived another lower bound *window-group distance* in Theorem 3 that can used together with deferred group subsequence retrieval. Using this group access mechanism, we finally proposed a further advanced algorithm *DeferredTopK* and proved its correctness in Theorem 4.

Through extensive experiments using both real and synthetic data sets, we showed that a substantial number of candidates were pruned out with the lower bounds in Theorems 1 and 3 and that many data pages were sequentially accessed with the deferred group subsequence retrieval in *DeferredTopK*. Extensive experiments showed that our advanced algorithms outperform competing algorithms including the sequential scan algorithm exploiting LB_Keogh by up to orders of magnitude. This speedup has been achieved by using aggressive pruning techniques based on the lower bounds and the deferred group subsequence retrieval. Overall, we believe our ranked subsequence matching methods provide comprehensive insight and a substantial framework for future research.

7. REFERENCES

[1] Agrawal, R., Faloutsos, C., and Swami, A., "Efficient Similarity Search in Sequence Databases," In *FODO*, 1993.

[2] Beckmann, N. et al., "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles," In *SIGMOD*, pp. 322-331, 1990.

[3] Berchtold, S., Bohm, C., and Kriegel, H.-P., "The Pyramid-Technique: Towards Breaking the Curse of Dimensionality," In *SIGMOD*, pp. 142-153, 1998.

[4] Berndt, D. and Clifford, J., "Finding Patterns in Time Series: a Dynamic Programming Approach," In *Advances in Knowledge Discovery and Data Mining,* pp. 229-248, AAAI/MIT, 1996.

[5] Bartolini, I., Ciaccia, P., and Patella, M. "WARP: Accurate Retrieval of Shapes Using Phase of Fourier Descriptors and Time Warping Distance," *IEEE PAMI*, pp. 142-147, 2005.

[6] Chan, K.-P. et al., "Haar Wavelets for Efficient Similarity Search of Time-Series: With and Without Time Warping," *IEEE TKDE*, pp. 686-705, 2003.

[7] Faloutsos, C., Ranganathan, M., and Manolopoulos, Y., "Fast Subsequence Matching in Time-Series Databases," In *SIGMOD*, pp. 419-429, 1994.

[8] Han, W., Lee, J., and Moon, Y., "Fundamental Techniques for Ranked Subseqeunce Matching," http://www.hummingbase.com/subseqrank.pdf, 2007.

[9] Hjaltason, G. and Samet, H., "Ranking in Spatial Databases," In *SSD*, pp. 83-95, 1995.

[10] Itakura, F., "Minimum Prediction Residual Principle Applied to Speech Recognition," *IEEE Trans. on ASSP*, Vol. ASSP-23, No. 1, pp. 67-72, 1975.

[11] Keogh, E. et al., "Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases," In *SIGMOD*, pp 151- 162, 2001.

[12] Keogh, E., "Exact indexing of dynamic time warping," In *VLDB*, pp 406-417, 2002.

[13] Keogh, E. "A Decade of Progress in Indexing and Mining Large Time Series Databases," In *VLDB*, Tutorial, 2006.

[14] Keogh, E., The UCR Time Series Data Mining Archive, http://www.cs.ucr.edu/∼eamonn/TSDMA/index.html.

[15] Kim, S.-W. et al., "An Index-Based Approach for Similarity Search Supporting Time Warping in Large Sequence Database" In *ICDE*, pp. 607-614, 2001.

[16] Lim, S.-H. et al., "Using Multiple Indexes for Efficient Subsequence Matching in Time-Series Databases," In *DASFAA*, pp. 65-79, 2006.

[17] Loh, W.-K. et al., "A Subsequence Matching Algorithm that Supports Normalization Transform in Time-Series Databases," *DMKD*, Vol. 9, No. 1, pp. 5-28, July 2004.

[18] Moon, Y.-S., Whang, K.-Y., and Loh, W.-K., "Duality-Based Subsequence Matching in Time-Series Databases," In *ICDE*, pp. 263-272, 2001.

[19] Moon, Y.-S., Whang, K.-Y., and Han, W.-S., "General Match: A Subsequence Matching Method in Time-Series Databases Based on Generalized Windows," In *SIGMOD*, pp. 382-393, 2002.

[20] Rafiei, D. et al., "Querying Time Series Data Based on Similarity," *IEEE TKDE*, Vol. 12, No. 5, 2000.

[21] Rabiner, L and Juang, B., Fundamentals of Speech Recognition, Englewood Cliffs, N. J., 1993.

[22] Ramakrishnan, R. and Gehrke, J., Database Management Systems, McGraw-Hill, 2000.

[23] Roussopoulos, N., Kelley, S., and Vincent, F., "Nearest Neighbor Queries," In *SIGMOD*, pp. 71-79, May. 1995.

[24] Sakoe, H. and Chiba, S, "Dynamic Programming Algorithm Optimization for Spoken Word Recognition," *IEEE Trans. on ASSP*, Vol. ASSP-26, No. 1, pp. 43-49, Feb. 1978.

[25] Seidl, T. and Kriegel, H.-P., "Optimal Multi-Step k-Nearest Neighbor Search," In *SIGMOD*, pp. 154-165, June. 1998.

[26] Sobell, M. O., Practical Guide to Linux Commands, Editors, and Shell Programming, Prentice Hall, 2005.

[27] Weber, R. et al., "A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces," In *VLDB*, pp. 194-205, pp. 194-205, Aug. 1998.

[28] Wong, T. and Wong, M, "Efficient Subsequence Matching for Sequences Databases under Time Warping," In *IDEAS*, pp. 139-138, 2003.

[29] Yi, B.-K., Jagadish, H. V., and Faloutsos, C., "Efficient Retrieval of Similar Time Sequences Under Time Warping," In *ICDE*, pp. 201-208, 1998.

[30] Yi, B.-K. and Faloutsos, C., "Fast Time Sequence Indexing for Arbitrary Lp Norms," In *VLDB*, pp. 385-394, 2000.

[31] Zhu, Y. and Shasha, D., "Warping Indexes with Envelope Transforms for Query by Humming," In *SIGMOD*, 2003.