Available online at www.sciencedirect.com

ScienceDirect

Information Sciences 178 (2008) 1461–1478

INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL

www.elsevier.com/locate/ins

Similar sequence matching supporting variable-length and variable-tolerance continuous queries on time-series data stream

Hyo-Sang Lim [a], Kyu-Young Whang [a,*], Yang-Sae Moon [b]

[a] *Department of Computer Science, Korea Advanced Institute of Science and Technology (KAIST), 373-1 Guseong-dong, Yuseong-gu, Daejeon 305-701, Republic of Korea*

[b] *Department of Computer Science, Kangwon National University, 192-1, Hyoja2-dong, Chunchon, Kangwon 200-701, Republic of Korea*

Received 31 October 2006; received in revised form 7 August 2007; accepted 21 October 2007

Abstract

We propose a new similar sequence matching method that efficiently supports variable-length and variable-tolerance continuous query sequences on time-series data stream. Earlier methods do not support variable lengths or variable tolerances adequately for continuous query sequences if there are too many query sequences registered to handle in main memory. To support variable-length query sequences, we use the *window construction mechanism* that divides long sequences into smaller windows for indexing and searching the sequences. To support variable-tolerance query sequences, we present a new notion of *intervaled sequences* whose individual entries are an interval of real numbers rather than a real number itself. We also propose a new similar sequence matching method based on these notions, and then, formally prove correctness of the method. In addition, we show that our method has the *prematching* characteristic, which finds future candidates of similar sequences in advance. Experimental results show that our method outperforms the naive one by 2.6–102.1 times and the existing methods in the literature by 1.4–9.8 times over the entire ranges of parameters tested when the query selectivities are low (<32%), which are practically useful in large database applications.
© 2007 Published by Elsevier Inc.

Keywords: Similar sequence matching; Time-series data; Data streams; Continuous queries

1. Introduction

A time-series is a sequence of real numbers representing values at specific time points [8,9,18]. Examples of time-series data include stock prices, weather data, network traffic data, and sensor data. There have been a number of efforts to handle the time-series data stored in databases [1,8,17,18,21,27]. Recently, the *data stream* has become of growing importance with new requirements due to advances in network technology and

[*] Corresponding author. Tel.: +82 42 869 5562; fax: +82 42 867 3562.
E-mail addresses: hslim@mozart.kaist.ac.kr (H.-S. Lim), kywhang@mozart.kaist.ac.kr (K.-Y. Whang), ysmoon@kangwon.ac.kr (Y.-S. Moon).

0020-0255/$ - see front matter © 2007 Published by Elsevier Inc.
doi:10.1016/j.ins.2007.10.026

mobile/sensor devices in emerging ubiquitous environments [6,9,10,14,20]. A data stream is a sequence of data entries that continuously arrive in a sequential order [2,19]. Examples include real-time sensor data, frequently changing trajectories of moving objects, and continuous flows of network packet data. The primary characteristic of a data stream is that the data are generated continuously, rapidly, unboundedly, and in real-time. Due to this characteristic, the entire data cannot be stored in a database, but should be processed on the fly. Hence, queries on data streams are not one-time queries, which are executed only once against stored data, but they are *continuous queries* that are registered in advance and run repeatedly over a period of time [2,19]. Thus, query processing in data streams can be seen as dual of that in databases because the former searches against stored continuous queries for data entries that newly arrive while the latter searches against stored data for one-time queries [5,15].

We define the time-series data that arrive in the form of data streams as the *data stream sequence* and the time-series data registered in the database as the *continuous query sequence*. We then define the *similar sequence matching on data stream* as finding the continuous query sequences that match the incoming data stream sequence up to a specific point in time within the user-specified tolerance.

We focus on similar sequence matching that can support a large number of variable-length and variable-tolerance continuous query sequences. Query sequences can be registered by many different users with different requirements on the lengths and tolerances. Nevertheless, existing results reported in the literature either support only fixed-length or fixed-tolerance continuous query sequences [9,14] or are unable to support a large number of query sequences with variable lengths or variable tolerances [10]. Other recent similar sequence matching methods reported in the data stream environment are only capable of handling one continuous query sequence [6,20].

We propose a new similar sequence matching method on data streams, which we call *Similar Sequence Matching based on Intervaled Sequence* (SSM-IS), that efficiently supports a large number of variable-length and variable-tolerance continuous query sequences. First, to support variable tolerances, we propose a new notion of the *intervaled sequence*. The intervaled sequence is defined as a sequence whose individual entries are an interval of real numbers rather than a real number itself. Using this notion, SSM-IS models a pair ⟨query sequence, tolerance⟩ as an intervaled sequence. Thus, it can efficiently support variable-tolerance query sequences by indexing and searching query sequences together with tolerances. We note that the work by Gao et al. [10] cannot support variable-tolerance continuous query sequences since it represents query sequences and tolerances separately. Next, to support variable-length continuous query sequences, we employ the *window construction mechanism* used in the traditional time-series subsequence matching methods [8,17,18]. The window construction mechanism divides long sequences into smaller windows. These divided windows are then used for indexing and searching.

We also use prematching and early abandoning [13] in SSM-IS. *Prematching* is a novel technique that finds not only current candidates, which can be concluded as similar sequences, but also *precandidates*, which are future candidates, whenever a new data entry arrives. We can use prematching to efficiently process similar sequence matching by reading precandidates and computing their distances in advance. We also use *early abandoning*, originally proposed by Keogh et al. [13], to reduce needless distance computation by computing the intermediate distance incrementally whenever a new data entry arrives and by abandoning as early as possible the candidates that cannot possibly be concluded to be similar sequences.

The rest of this paper is organized as follows. In Section 2, we introduce the data stream sequence and continuous query sequence, and then, formally define the problem of similar sequence matching on data streams. In Section 3, we review previous work on similar sequence matching on data streams. In Section 4, we propose a new model of representing continuous query sequences for performing similar sequence matching, and then, present the notion of prematching in the model. In Section 5, we present SSM-IS, the similar sequence matching method that supports variable-length and variable-tolerance continuous query sequences. In Section 6, we present the results of performance evaluation. Finally, in Section 7, we summarize and conclude the paper.

2. Preliminaries

In this section, we formally define the data stream sequence, continuous query sequence, and the similar sequence matching problem. We first summarize in Table 1 the notation to be used throughout the paper.

Table 1
Summary of notation

Symbols	Definitions
N	Number of continuous query sequences
Q_k	The kth continuous query sequence $(1 \leqslant k \leqslant N)$
ϵ_k	User-specified tolerance of continuous query sequence Q_k (maximum allowable distance between two sequences)
D	A data stream
$\text{Len}(X)$	Length of sequence X
$X[i]$	The ith entry of sequence $X(1 \leqslant i \leqslant \text{Len}(X))$
$X[i:j]$	Subsequence of X, including entries from the ith one to the jth $(i < j)$
$dist(X, Y)$	Euclidean distance between sequences X and Y $(\text{Len}(X) = \text{Len}(Y))$
L	Base length of sequences for variable lengths (see Section 6)
s	Base selectivity of query sequences for variable tolerances (see Section 6)

2.1. Data stream sequences and continuous query sequences

The *data stream sequence* is a time-series data stream to which new data entries are continually appended every time unit. When D is an infinite data stream, the data stream sequence at the time point t can be represented as in Eq. (1) using the notation in Table 1:

$$D[1:t] = \{D[1], D[2], \ldots, D[t]\}, \quad D[i] = \text{the time-series data entry at time point } i \quad (1 \leqslant i \leqslant t). \tag{1}$$

That is, similar sequence matching on the data stream D deals with the dynamic data stream sequence $D[1:t]$ at the time point t, which is continuously changing as time passes, instead of the static sequences stored in a database.

The continuous query sequence is a sequence of time-series data registered by the users to detect occurrence of similar sequences in a data stream. (Hereafter, we call a continuous query sequence as a *query sequence* unless there is confusion.) In this paper, we deal with the case where each query sequence may have its own length and tolerance (i.e., variable lengths and variable tolerances) according to individual users' interests. A pair of the kth registered query sequence Q_k and its tolerance ϵ_k can be represented as in Eq. (2):

$$(Q_k, \epsilon_k) = (\{Q_k[1], Q_k[2], \ldots, Q_k[\text{Len}(Q_k)]\}, \epsilon_k). \tag{2}$$

2.2. Similar sequence matching on data streams

Similar sequence matching in time-series databases is a problem of finding data sequences similar to the given query sequence from the database [1,17]. In this paper, we use the similarity model based on the Euclidean distance $dist(X, Y)(= \sqrt{\sum_{i=1}^{n}(X[i] - Y[i])^2})$ between two sequences $X(= \{X[1], X[2], \ldots, X[n]\})$ and $Y(= \{Y[1], Y[2], \ldots Y[n]\})$ of the same length n.[1]

We say two sequences X and Y are *similar* if the distance $D(X, Y)$ is less than or equal to the user-specified tolerance ϵ. More specifically, we define that two sequences X and Y are in ϵ-*match* if $dist(X, Y)$ is less than or equal to ϵ [17].

Similar sequence matching on data streams differs from the traditional one as follows. Similar sequence matching on data streams deals with continually changing sequences as targets of the matching, while the traditional matching deals with static data sequences stored in the database. Similar sequence matching on data streams finds similar query sequences by comparing the incoming data stream sequence with stored query sequences at every time point as new data entries arrive. Thus, similar sequence matching on data stream is formally defined as follows [9]:

[1] Our similarity model can be easily extended to support an arbitrary L_p-norm (the Euclidian distance used in the paper is L_2-norm). L_p-norm is a well known similarity model and covers a wide range of applications [26]. Besides L_p-norms, there are several distance metrics such as the time-warping distance [25,27] and the longest common subsequence [11]. In this paper, we focus on L_p-norms and briefly discuss about supporting other similarity models at the end of Section 5.

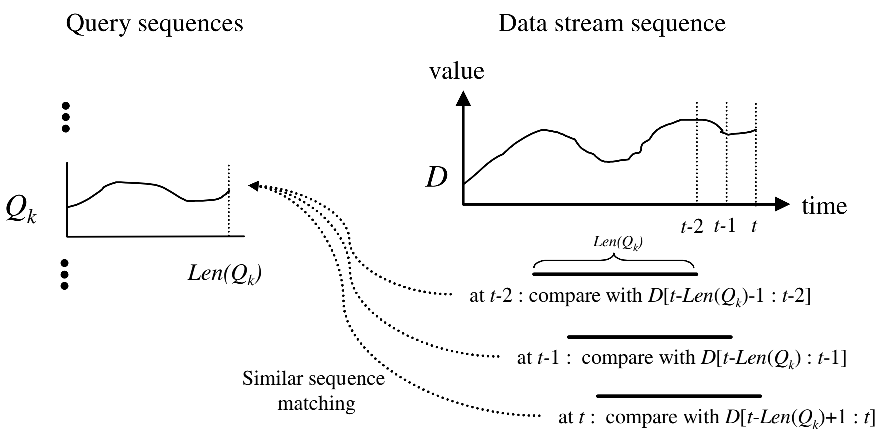

Query sequences
Data stream sequence
value
Q_k
$Len(Q_k)$
D
time
t-2 t-1 t
$Len(Q_k)$
at t-2 : compare with $D[t\text{-}Len(Q_k)\text{-}1 : t\text{-}2]$
at t-1 : compare with $D[t\text{-}Len(Q_k) : t\text{-}1]$
at t : compare with $D[t\text{-}Len(Q_k)\text{+}1 : t]$
Similar sequence matching

Fig. 1. An example of the similar sequence matching process on data streams.

Definition 1 [9]. Let D be a data stream, and $\{Q_k | 1 \leqslant k \leqslant N\}$ be a set of query sequences registered in the system. *Similar sequence matching on D* is the problem of finding query sequences Q_k where $D[t - \text{Len}(Q_k) + 1 : t]$ and Q_k are in ϵ-match at every time point t.

Fig. 1 shows an example of the similar sequence matching process where the query sequence Q_k is compared with the data stream D from time point $t - 2$ to t. In the figure, $D[t - \text{Len}(Q_k) - 1 : t - 2]$ is compared with Q_k at the time point $t - 2$, $D[t - \text{Len}(Q_k) : t - 1]$ at $t - 1$, and $D[t - \text{Len}(Q_k) + 1 : t]$ at t, respectively. That is, whenever a new data entry arrives as time passes, the distance comparison will be done between query sequences and the most recent data stream sequence that contains the new entry. Hereafter, we call this data stream sequence as a *current data sequence*.

3. Related work

In this section, we review previous work on similar sequence matching on data streams. For similar sequence matching on static time-series databases, readers are referred to the literature for whole matching [1,26] and for subsequence matching [8,16,17].

Gao and Wang [9] proposed a similar sequence matching method (let us call it Gao-1) based on prediction. Gao-1 predicts future stream entries based on the previous entries that already arrived, and computes the distances between query sequences and the predicated data stream sequences in advance. Then, whenever an actual entry arrives, it identifies similar sequences by only adjusting the prediction errors. This method has an advantage of reducing CPU cost since it uses Fast Fourier Transform (FFT) to do distance computation for many predicted sequences in a batch manner rather than one by one. However, it also has disadvantages: (1) it has an overhead of adjusting the prediction errors. This overhead is especially big when the difference between the predicted entry and the actual one is large; (2) it is difficult to support a large number of query sequences since the distances for all query sequences should be computed at each time point.

Gao et al. [10] proposed another similar sequence matching method (let us call it Gao-2) based on prefetching. This method solves the k-Nearest Neighbor (k-NN) problem that finds the most similar k query sequences against the current data sequence. The k-NN problem is different from similar sequence matching in that the number k of results is given instead of tolerance ϵ. However, it is analogous to similar sequence matching in that it finds similar sequences, i.e., finds sequences close in distance to the query sequences. Gao-2 stores a large number of query sequences in disk, transforms them into lower-dimensional points, and stores the points in a multidimensional index. Then, whenever a new data entry arrives, k nearest query sequences are found using this index for the current data sequence starting from the new data entry. Gao-2 uses a prefetching strategy that predicts the very next entry using the previous ones and retrieves the query sequences, which are potentially k-NN candidates in the near future. Unlike Gao-1, Gao-2 is capable of supporting a large number of query sequences by using the multidimensional index and by amortizing disk accesses through the prefetch-

Table 2
Comparison of the proposed method and the other methods in the literature

	Gao-1 [9]	Gao-2 [10]	MSM [14]	Our method
Supporting variable-length query sequences	○	×	×	○
Supporting variable-tolerance query sequences	○	×	×	○
Supporting a large number of query sequences	×	○	○	○

ing strategy. However, it also has a drawback that only fixed-length and fixed-tolerance query sequences are allowed.

Lian et al. [14] proposed a similar sequence matching method for multiple time-series data streams. For efficient processing, this method uses a new approximation technique, the multi-scaled segment mean(MSM), instead of lower-dimensional transformation. MSM is suitable in a data stream environment since it can be incrementally calculated by one pass scan over input data elements. Based on MSM, the paper also proposed a multi-step filtering technique that progressively eliminates false alarms from a rough approximation level with low cost to a fine approximation level with high cost. The technique has the desirable property of eliminating false alarms in data streams at the early stage of filtering with low cost. However, it also has a drawback that only fixed-length and fixed-tolerance query sequences are allowed.

Recently, other research results on similar sequence matching in data streams have been reported in the literature. Sakurai et al. [20] proposed a similar sequence matching method based on the time-warping distance. Chen et al. [6] proposed a new distance model called SpADe and provided a similar sequence matching method based on the model. However, all the methods proposed are capable of handling only one continuous query sequence, and cannot be easily extended to supporting a large number of continuous query sequences.

Supporting variable-length and variable-tolerance query sequences while supporting a large number of query sequences is the major contribution of our proposed method compared with earlier ones in the literature. In this paper, we compare our method with Gao-1, Gao-2, and MSM, which are capable of supporting multiple query sequences. Table 2 summaries the differences among our method and the earlier ones.

4. Modeling similar sequence matching on data streams

4.1. Modeling continuous query sequences

To support variable tolerances, we need new models of query sequences and of similar sequence matching. In earlier work in the literature [8,10,17], only query sequences are indexed. Tolerances are independently used to construct the region queries when searching the index; i.e., the tolerance is modeled as part of search operations rather than as part of the index. This approach is reasonable when supporting only fixed tolerances, i.e., when all query sequences have the same tolerance. However, it is not appropriate when supporting variable tolerances. To support variable tolerances in earlier work, the maximum value of a large number of different tolerances should be used to construct range queries. This will degrade the performance since it causes a lot of false alarms. Furthermore, this problem is aggravated as the number of registered query sequences increases since the differences among tolerances becomes larger.

To properly support variable tolerances, i.e., to allow each query sequence to have its own tolerance, we propose a new notion of the *intervaled sequence*. In the intervaled sequence, each entry is an interval of real numbers. We transform the original query sequences to the intervaled sequences using the tolerance of each entry as the interval of the transformed entry. That is, we model a query sequence and its tolerance together by storing the intervaled sequence. We formally define the intervaled sequence in Definition 2. In Definition 3, we also define *δ-interval match* that represents inclusion relationship between a sequence and an intervaled sequence.

Definition 2. For a sequence X with the interval δ, the *intervaled sequence* X denoted by X^{δ} is defined as the sequence where each real number entry $X[i]$ is replaced by the interval $[X[i]-\delta, X[i]+\delta]$.

Definition 3. For two sequences X and Y of the same length and an interval δ, the sequence Y is in *δ-interval match* with X^{δ} if $X[i]-\delta \leqslant Y[i] \leqslant X[i]+\delta$ for every i, $1 \leqslant i \leqslant \mathrm{Len}(X)$.

Fig. 2 shows an example of intervaled sequences and δ-interval match. Fig. 2(a) shows the original sequences X and Y; Fig. 2(b) shows the intervaled sequence X^{δ} and the sequence Y that is in δ-interval match with X^{δ}.

In the view point of using tolerances, our similar sequence matching method is dual to the method in the time-series database. The latter uses a tolerance as the range of a region query representing a query sequence, and then, finds indexed data sequences that are within the range [1,18]. In contrast, the former uses tolerances as ranges of indexed regions representing query sequences, and then, finds the regions whose ranges include a data sequence. In other words, the former uses a tolerance as the range of a region query and the latter as that of an indexed region object. Therefore, in the latter, we can reasonably expect a similar degree of false alarms to that in the former.

4.2. Modeling similar sequence matching

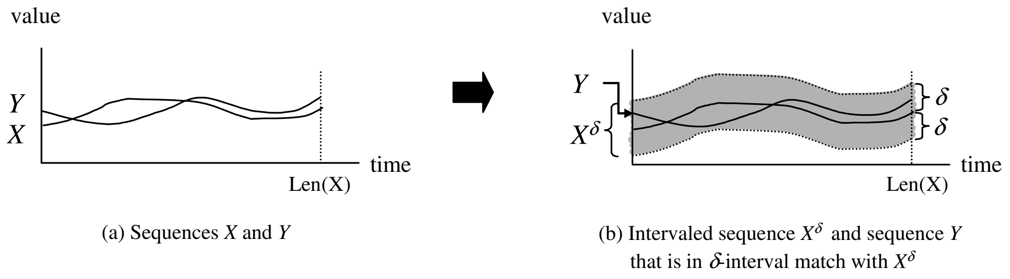
For ease of exposition, we first explain how to support variable tolerances for fixed-length query sequences using intervaled sequences, and then, expand this idea to support variable-length query sequences. If all the query sequences are of the same length, similar sequence matching can be done by using the intervaled sequence whose interval δ is set to the tolerance ϵ. That is, for each query sequence with tolerance ϵ, we construct the intervaled sequence by setting the interval δ to the tolerance ϵ. Then, we perform the similar sequence matching by finding the intervaled query sequences that are in δ-interval match with the current data sequence. Likewise, by constructing the intervaled sequences that have different intervals for different query sequences, we can support variable tolerances. Lemma 1 shows correctness of this approach.

Lemma 1. *For two sequences X and Y of the same length, if X and Y are in ϵ-match, then the sequence Y is in δ-interval match with the intervaled sequence X^{δ}, where $\delta = \epsilon$. That is, the following Eq. (3) holds:*

$$dist(X,Y) \leqslant \epsilon \Rightarrow \bigwedge_{i=1}^{\mathrm{Len}(X)} (X[i] - \delta \leqslant Y[i] \leqslant X[i] + \delta). \tag{3}$$

Proof. See Appendix A. □

To support variable-length query sequences, we employ the window construction mechanism used in the traditional time-series subsequence matching methods [8,17,18]. The window construction mechanism divides long sequences into smaller windows. These divided windows are then used for indexing and searching. As subsequence matching methods in time-series databases, FRM by Faloutsos et al. [8] and DualMatch by Moon et al. [17] have been proposed. In this paper, we use the window construction mechanism of DualMatch. We divide stored sequences (i.e., query sequences) into *disjoint windows* of size ω and data stream sequences into *sliding windows* of size ω. That is, for every time point t, we construct a sliding window $D[t - \omega + 1 : t]$ containing the entry $D[t]$ that newly arrives and perform the similar sequence matching with this sliding window.



(a) Sequences X and Y (b) Intervaled sequence X^{δ} and sequence Y that is in δ-interval match with X^{δ}

Fig. 2. An example of intervaled sequences and δ-interval match.

We now propose a new similar sequence matching method, which we call *Similar Sequence Matching based on Intervaled Sequence* (SSM-IS), based on the notion of intervaled sequences and the window construction mechanism. SSM-IS consists of index building and matching steps as follows:

- Index building step: For each query sequence Q_k, (1) an intervaled sequence Q_k^δ is constructed; and (2) Q_k^δ is divided into disjoint intervaled windows of size ω; and (3) each intervaled window is stored in a multidimensional index as a region (we note that an intervaled window of size ω is represented as an ω-dimensional *region* while an original window of size ω as an ω-dimensional *point*).
- Matching step: When a new data entry arrives into the data stream D at the time point t, (1) a sliding window $D[t-\omega+1:t]$ is constructed; (2) the intervaled windows that are in δ-interval match with the sliding window are retrieved from the index, forming a candidate set (we note that these windows can be easily retrieved by evaluating the point query that finds regions including the point representing the sliding window); and (3) similar sequences are identified through post-processing [1,8] eliminating false alarms [8,17].

To guarantee correctness of SSM-IS, we first determine an appropriate interval δ used to construct the intervaled sequence Q_k^δ for each query sequence Q_k. That is, we determine the interval δ so as to guarantee that no false dismissal occur. Theorem 1 shows the theoretical rationale for determining the interval δ.

Theorem 1. *For a given query sequence Q_k with tolerance ϵ_k, SSM-IS incurs no false dismissal if $\delta = \epsilon_k/\sqrt{p}$. Here, $p = \lfloor \mathrm{Len}(Q_k)/\omega \rfloor$, and ω is the window size.*

Proof. We simply prove it by using the theorems by Faloutsos et al. [8]. For sequences S_1 and S_2 of the same length, the following Eq. (4) holds [8]:

$$dist(S_1,S_2) \leqslant \epsilon \;\Rightarrow\; dist(S_1[l:m],S_2[l:m]) \leqslant \epsilon \quad (1 \leqslant l < m \leqslant \mathrm{Len}(S_1)). \tag{4}$$

Eq. (4) indicates that, when Q_k is divided into disjoint windows, we can ignore the remainder $Q_k\left[\lfloor\frac{\mathrm{Len}(Q_k)}{\omega}\rfloor\cdot\omega+1:\mathrm{Len}(Q_k)\right]$, those whose length is less than ω, without causing any false dismissal [17]. Thus, we assume that a query sequence is composed of exactly p disjoint windows. Then, for the jth disjoint window $DW_j = Q_k[(j-1)\cdot\omega+1:j\cdot\omega]$ $(1 \leqslant j \leqslant p)$ in Q_k and the corresponding sliding window $SW_j = D[(t-\mathrm{Len}(Q_k)+1)+((j-1)\cdot\omega):(t-\mathrm{Len}(Q_k)+1)+(j\cdot\omega-1)]$ in $D[t-\mathrm{Len}(Q_k)+1:t]$, the following Eq. (5) holds [8]:

$$dist(Q_k, D[t-\mathrm{Len}(Q_k)+1:t]) \leqslant \epsilon_k \;\Rightarrow\; \bigvee_{j=1}^{p} dist(DW_j, SW_j) \leqslant \epsilon_k/\sqrt{p}. \tag{5}$$

By Lemma 1, the right side of Eq. (5), $dist(DW_j, SW_j) \leqslant \epsilon_k/\sqrt{p}$, means that the intervaled window DW_j^δ is in δ-interval match with SW_j where $\delta = \epsilon_k/\sqrt{p}$. Thus, the query sequence Q_k whose intervaled window DW_j^δ is in δ-interval match with the sliding window SW_j is treated as a candidate, then false dismissal does not occur since the necessary condition of Eq. (5) is satisfied. Consequently, SSM-IS does not incur any false dismissal. □

Example 1. Fig. 3 shows an example of similar sequence matching for two query sequences Q_1 and Q_2 using SSM-IS. The query sequences have different lengths and different tolerances ϵ_1 and ϵ_2. By Theorem 1, in SSM-IS, we construct the intervaled query sequences $Q_1^{\delta_1}$ and $Q_2^{\delta_2}$, where $\delta_1 = \epsilon_1/\sqrt{p_1}$ and $\delta_2 = \epsilon_2/\sqrt{p_2}$. Then, we divide $Q_1^{\delta_1}$ and $Q_2^{\delta_2}$ into disjoint windows of size ω and store these disjoint windows in the multidimensional index as regions. When a new data entry arrives at the time point t, we construct sliding window $SW_t(=D[t-\omega+1:t])$ of size ω. Next, by searching the multidimensional index, we retrieve the candidate sequences that are in δ-interval match with the sliding window, i.e., those whose regions include the point representing the sliding window. In Fig. 3, query sequence Q_1 is determined as a candidate since the region represented by the intervaled window $DW_2^{\delta_1}$ contains the sliding window SW_t.

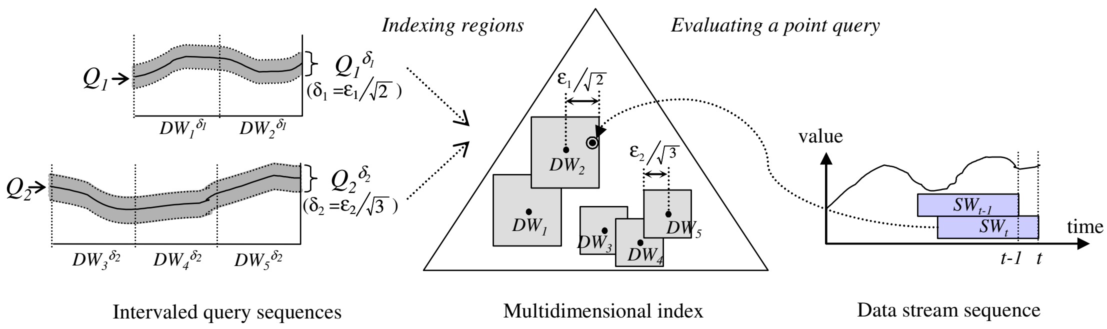


Fig. 3. An example of similar sequence matching for variable-length and variable-tolerance query sequences in SSM-IS.

4.3. Prematching and early abandoning

According to Theorem 1, at every time point when a new data entry arrives, we are able to find the future candidates of similar sequences as well as the current candidates. This is possible because a query sequence can be divided into more than one disjoint window. Suppose an intervaled query sequence is divided into two or more disjoint windows, and a disjoint window except the last one is in δ-interval match with the sliding window of the data stream. Then, we can retrieve the query sequence as a candidate by searching the index at the current time point, but we can determine whether the query sequence is a similar sequence or not only when the rest of the future data entries arrive. Thus, we regard these query sequences as *future* candidates. To further illustrate, suppose D is a data stream, Q and Q^δ are a query sequence and its intervaled sequence, and Q is divided into p disjoint windows. At the time point t, if the sliding window $D[t-\omega+1:t]$ is in δ-interval match with the pth (i.e., the last) window $Q^\delta[(p-1)\cdot\omega+1:p\cdot\omega]$ of Q^δ, then we determine Q as a candidate and immediately identify whether the candidate is a similar sequence or a false alarm by computing the distance between Q and $D[t-\mathrm{Len}(Q)+1:t]$. However, if $D[t-\omega+1:t]$ is in δ-interval match with the ith window $Q^\delta[(i-1)\cdot\omega+1:i\cdot\omega]$ where i is less than p $(1 \leqslant i < p)$, then we regard Q as a candidate that is potentially in δ-interval match with $D[t-\mathrm{Len}(Q)+i+1\cdot\omega:t+(p-i)\cdot\omega]$ at the future time point $t+(p-i)\cdot\omega$. Thus, the query sequence is a future candidate that will be post-processed at a future time point. The important observation is that we can find future candidates at the current time point in advance. We call this technique of finding future candidates as *prematching* and the future candidates retrieved by prematching as *precandidates*. Prematching can be used to reduce the response time of similar sequence matching by reading precandidates and computing their distances in advance.

The following Corollary 1 derived from Theorem 1 shows when we can identify a precandidate, which is retrieved from the index at the time point t, as a similar sequence by performing post-processing. Using Corollary 1, we know that the time point of identifying similar sequences will be determined by the position of the intervaled window that is in δ-interval match with the sliding window at the time point t.

Corollary 1. *For a given query sequence Q with tolerance ϵ and the current time point t, if the sliding window $D[t-\omega+1:t]$ of data stream D is in δ-interval match with the ith $(1 \leqslant i \leqslant p)$ disjoint window $Q^\delta[(i-1)\cdot\omega+1:i\cdot\omega]$ of Q^δ, then Q becomes a precandidate that will be concluded as a similar sequence or a false alarm at the time point $t+\mathrm{Len}(Q)-\omega\cdot i$. Here, $p=\lfloor \mathrm{Len}(Q)/\omega \rfloor$ and $\delta=\epsilon/\sqrt{p}$.*

Proof. See Appendix B. □

Example 2. Fig. 4 shows an example of prematching and precandidates. Two intervaled query sequences $Q_1^{\delta_1}$ and $Q_2^{\delta_2}$ are divided into three intervaled windows of size ω. At the time point t, Q_1 and Q_2 are determined as candidates since the intervaled window $DW_3^{\delta_1}$ of $Q_1^{\delta_1}$ and $DW_5^{\delta_2}$ of $Q_2^{\delta_2}$ are in δ-interval match with the sliding window $D[t-\omega+1:t]$. Here, Q_1 can be concluded as a similar sequence at the time point t because comparison of Q_1 and $D[t-3\omega+1:t]$ can be done at the time point t. However, Q_2 becomes a precandidate at the time point t because comparison of Q_2 and $D[t-2\omega+1:t+\omega]$ will be done at the time point $t+\omega$. Hence,

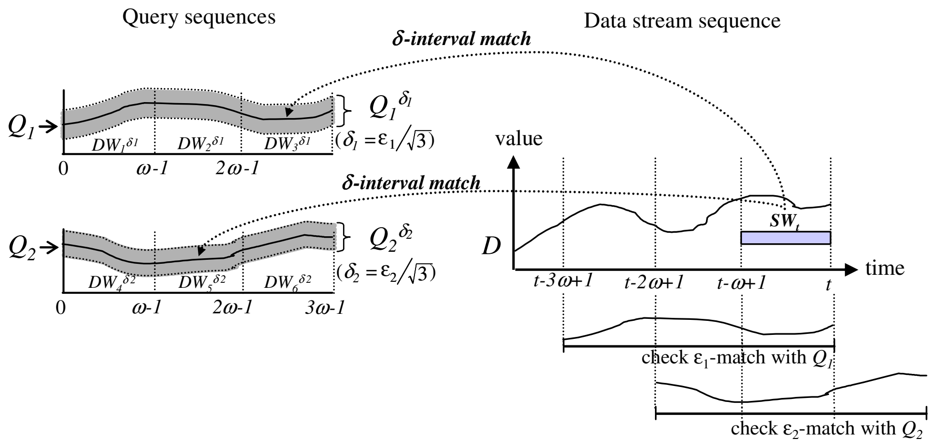

Query sequences
Data stream sequence
δ-interval match
Q_1
$Q_1^{\delta_1}$
$DW_1^{\delta_1}$ $DW_2^{\delta_1}$ $DW_3^{\delta_1}$
$(\delta_1 = \varepsilon_1/\sqrt{3})$
0 ω-1 2ω-1
δ-interval match
Q_2
$Q_2^{\delta_2}$
$DW_4^{\delta_2}$ $DW_5^{\delta_2}$ $DW_6^{\delta_2}$
$(\delta_2 = \varepsilon_2/\sqrt{3})$
0 ω-1 2ω-1 3ω-1
value
D
SW_t
time
t-3ω+1 t-2ω+1 t-ω+1 t
check ε_1-match with Q_1
check ε_2-match with Q_2

Fig. 4. An example of prematching and precandidates in SSM-IS.

the precandidate Q_2 cannot be concluded as a similar sequence at the time point t and must wait for the arrival of future data entries until the time point $t + \omega$.

To efficiently process similar sequence matching, we also use early abandoning [13]. *Early abandoning* computes the intermediate distance incrementally whenever a new data entry arrives and abandons as early as possible the precandidates that will not be concluded to be similar sequences. By using early abandoning, we can reduce needless distance computations for the precandidates whose distance already exceeded the given tolerance.

5. Similar sequence matching algorithm that supports variable-length and variable-tolerance query sequences

In this section, we propose index building and matching algorithms of SSM-IS. Fig. 5 shows an overview of similar sequence matching in SSM-IS. As mentioned in Section 4, SSM-IS consists of two steps: the index building step (① in Fig. 5) and the matching step (②–⑤). The index building step is executed only once before the matching step. Here, all the registered query sequences are stored into the multidimensional index. The matching step is executed whenever a new data entry arrives. Here, candidate query sequences that are potentially in ϵ-match with the data stream sequence are identified by searching the multidimensional index(②) and are read from the disk (③).[2] Then, similar sequences are identified by computing the distances between candidate query sequences and the data stream sequence to eliminate false alarms (*refinement* ④). The final results are returned to the users (⑤).

Fig. 6 shows the index building algorithm *BuildIndex*. The input to the algorithm is a database containing continuous query sequences with tolerances. The output is a multidimensional index, which will be used in similar sequence matching. Here, the window size ω for dividing query sequences into disjoint windows is set to the minimum length of the registered query sequences. The window size must be less than or equal to the length of a query sequence to guarantee correctness of the matching algorithm while it should be maximized to reduce false alarms [17].[3] See the literature [17] for the relationship between the window size and the number of false alarms in detail.

[2] These candidate query sequences are managed in the main memory to efficiently process similar sequence matching. This is feasible since the number of candidates is much smaller than that of total query sequences in the case of low selectivity [8], which is generally more useful case than high selectivity [16,18].

[3] If the lengths of query sequences have a large variation, the window size ω becomes smaller than the average query length. Then, performance of SSM-IS degrades since more false alarms occur due to a smaller window size. This problem can be solved by grouping query sequences that are similar in length; by building a separate index for each group; by searching these indexes, respectively; and finally by merging the results from each index.

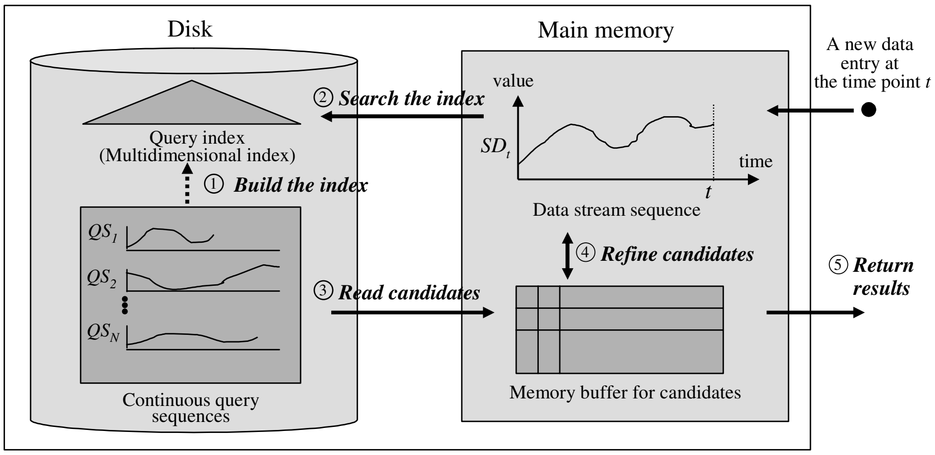


Fig. 5. Overview of similar sequence matching in SSM-IS.

Algorithm **BuildIndex**

```
Input:     Database $db$ that contains N query sequences with tolerances
Output:    An $f$-dimensional index $index$, which will be used for similar sequence matching
Algorithm:
1   Initialize the multidimensional index $index$
2   For each query sequence $Q$ with identifier $Qid$ and tolerance $\varepsilon$ in $db$, DO
    2.1 Divide $Q$ into $p$ disjoint windows $DW_j (1 \le j \le p)$ of size $\omega$
    2.2 For each $DW_j$ starting at $offset = (j-1) \cdot \omega + 1$, DO
        2.2.1  Transform the $\omega$-dimensional window $DW_j$ to an $f$-dimensional window $f\text{-}DW_j$ by using
               the lower-dimensional transformation.
        2.2.2  Construct the intervaled window $f\text{-}DW_j^{\delta}$ with interval $\delta = \varepsilon/\sqrt{p}$.
        2.2.3  Construct the record <$f\text{-}DW_j^{\delta}$, $Qid$, $offset$, $Len(Q)$>
        2.2.4  Insert the record, whose key is $f\text{-}DW_j^{\delta}$, into the $index$.
```

Fig. 6. The index building algorithm BuildIndex of SSM-IS.

In the algorithm BuildIndex, we first initialize a multidimensional index in Step 1 and then store query sequences into the index in Step 2. In Step 2.1, we divide each query sequence Q into disjoint windows of size ω. In Step 2.2, we store these disjoint windows into the multidimensional index. Specifically, in Step 2.2.1, we transform a disjoint window DW_j to an f-dimensional window $f\text{-}DW_j$ ($f \ll \omega$) by using the lower-dimensional transformation such as Discrete Fourier Transform and Wavelet Transform [7]. The reason for using the lower-dimensional transformation is that the performance of multidimensional indexes decreases as the number of dimensions increases (called *the dimensionality curse* [23]). In Step 2.2.2, by using Theorem 1, we construct an intervaled window $f\text{-}DW_j^{\delta}$ with $\delta = \epsilon/\sqrt{p}$ for the window $f\text{-}DW_j$. Here, the intervaled window is represented as a region in f-dimensional space. In Step 2.2.3, we construct a record consisting of the f-dimensional region $f\text{-}DW_j^{\delta}$, the query sequence identifier *Qid*, the start offset *offset* $(= (j-1) \cdot \omega + 1)$ of disjoint window DW_j, and the query sequence length Len(Q). Finally, in Step 2.2.4, we store this record into the index. When using the index for searching, the identifier *Qid* will be used to retrieve the candidate query sequence Q

from the database *db*; the start offset *offset* to find the location of the candidate sequence in the data stream *D*; the length Len(Q) to check whether the candidate query sequence is a candidate or a precandidate at the current time point.

Complexity of BuildIndex: The space and time complexities of BuildIndex depend on the multidimensional index structure used and the number of objects indexed. Here, we use the R^*-tree [3] as the index structure. Similar to the B$^+$-tree, the space and time complexities of the R^*-tree are known as O(n) and O($n \cdot \log n$) where n is the number of objects indexed. Since BuildIndex divides query sequences into disjoint windows of size ω and stores those windows into the index, the number of objects indexed is equal to that of disjoint windows. We let the number N_{dw} be $\sum_{i=1}^{N}\lfloor \mathrm{Len}(Q_i)/\omega \rfloor$. We then conclude the space complexity of BuildIndex is $\mathrm{O}(N_{dw})$ and the time complexity is $\mathrm{O}(N_{dw} \cdot \log N_{dw})$.

Fig. 7 shows the matching algorithm *Matching*. The inputs to the algorithm are a database *db* containing query sequences with tolerances, a multidimensional index *index*, a data stream *D*, and a set *candidate_set* of

```
Algorithm Matching
Input:  (1) Database db that contains N query sequences with tolerances
        (2) f-dimensional index index that has been created by the algorithm BuildIndex
        (3) Data stream D at the time point t
        (4) Set of precandidates candidate_set at the time point t-1 (when t=1, this is an empty set)
Output: (1) Similar sequences at the time point t (identifiers of query sequences)
        (2) Set of precandidates candidate_set at the time point t
Algorithm:
1 Index Searching Step: for the sliding window $SW_t$ (= $D[t-\omega+1,t]$) of D, DO
   1.1 Transform $SW_t$ into an f-dimensional point f-point by using a lower-dimensional transformation.
   1.2 Evaluate a point query against index to find regions that include f-point and insert the results
       with the time point match_time ( = t) and intermediate distance intermediate_dist(0 of now) into
       candidate_set.
2 Post-processing Step: for each record <$f\text{-}DW_j^{\delta}$, Qid, offset, Len(Q)> in candidate_set DO
   2.1 If the candidate query sequence Q with the identifier Qid can be finally identified as a similar sequence
       at the current time point t ( = match_time + Len(Q)– offset), i.e., Q is a candidate DO (refinement)
      2.1.1 If Q was not prefetched yet, read Q with its tolerance $\varepsilon$ from db.
      2.1.2 Compute the distance dist(Q, D[t –Len(Q)+1, t] ). Here, if an intermediate_dist has been
            computed up to the time point t-1, then compute the distance between Q[Len(Q)] and D[t] only
            and add it to the intermediate_dist (incremental distance calculation).
      2.1.3 If the distance is less than or equal to $\varepsilon$, return Qid as a result since Q is a similar
            sequence. Remove it from candidate_set. If the distance is larger than $\varepsilon$, remove Qid
            from candidate_set since it is a false alarm.
   2.2 If Q cannot be identified as a similar sequence at the time point t, i.e., Q is a precandidate, DO
      2.2.1 Read the query sequence Q and its tolerance $\varepsilon$ from db (prefetching).
      2.2.2 Compute the intermediate distance up to the time point t, intermediate_dist = dist(Q [1,
            offset+ $\omega$+ t–match_time–1], D[match_time– offset, t]). Here, if there is an intermediate_dist
            that has been computed at the time point t-1, then use the incremental distance calculation
            method.
      2.2.3 if intermediate_dist is larger than $\varepsilon$, remove Qid from candidate_set since it is a false
            alarm (early abandoning). If intermediate_dist is less than or equal to $\varepsilon$, update
            intermediate_dist in candidate_set.
```

Fig. 7. The matching algorithm Matching of SSM-IS.

precandidates constructed in the previous time points. The outputs are identifiers of query sequences that are in ϵ-match with the current data stream sequence and a set of precandidates updated at the current time point. Here, the size ω for sliding windows is set to the same value used for the disjoint windows in the algorithm BuildIndex. The algorithm Matching consists of two steps: the *index searching step* that finds candidate query sequences by searching the index (Step 1) and the *post-processing step* that identifies similar sequences through refinement from the candidates (Step 2).

In the index searching step, we find a set of candidates by searching the multidimensional index. In Step 1.1, we transform the sliding window SW_t consisting of the most recent ω data entries(current data sequence) to an f-dimensional point f-point by using the lower-dimensional transformation. In Step 1.2, we evaluate a point query against the multidimensional index to find regions that include f-point. Next, we construct the candidate set *candidate_set* each element of which consists of a retrieved record, the current time point *match_time* $(= t)$, and the intermediate distance value *intermediate_dist*. Here, *match_time* will be used to know the time point when the candidate can be concluded as a similar sequence. An *intermediate_dist* for each candidate query sequence is a partial distance that is computed by using the data entries that have arrived up to the current time point. That is, for each precandidate, the final distance cannot be computed until more data entries arrive. In the index searching step, however, each *intermediate_dist* is set to 0 as an initial value since the distance has not been computed yet.

In the post-processing step of the algorithm, we identify similar sequences at the time point t from candidate sequences. In Step 2.1, we perform post-processing (i.e., refinement) for the candidate query sequences, which could be identified to be a similar sequence at the current time point. In Step 2.1.1, if the query sequence has not been prefetched yet, we read the sequence from the database *db*. In Step 2.2.2, we compute the distance between the query sequence retrieved and the current data sequence. Here, if there is already an intermediate distance *intermediate_dist* computed for the query sequence in advance, we do incremental distance calculation, where we compute the final distance by only computing the distance between a new data entry $D[t]$ and the last entry of the query sequence $Q[\text{Len}(Q)]$ reflecting to *intermediate_dist*. In Step 2.1.3, we remove false alarms returning only those query sequences in ϵ-match with the current data sequence as the results. In Step 2.2, we perform post-processing for a precandidate, which cannot be identified to be a similar sequence at the time point t. In particular, in Step 2.2.3, we may abandon as false alarms some precandidates early whose *intermediate_dist* already exceeds ϵ (*early abandoning*).

Complexity of Matching: The space and time complexities of Matching depend on various factors such as the selectivity of query sequences and the number of candidates. For convenience, we analyze these complexities in the worst case. First, we analyze the space complexity. Here, we need to consider the space for storing and indexing query sequences and that for maintaining *candidate_set*. The space for storing query sequences is $N \cdot (\text{Max}(\text{Len}(Q_i)))$ and that for indexing is N_{dw} as we previously analyzed for BuildIndex. In the worst case, the space for *candidate_set* at the time point t is $N \cdot (\text{Max}(\text{Len}(Q_i)) - \omega)$ since every query sequence may be selected as a candidate at every time point from $t - \text{Max}(\text{Len}(Q_i))$ to t. Therefore, the space complexity becomes $\text{O}(N_{dw} + N \cdot \text{Max}(\text{Len}(Q_i)))$. Next, we analyze the time complexity. Here, we need to consider the time for searching the index and updating *candidate_set*. The time for searching the index (i.e., R^*-tree) is $\text{O}(\log N_{dw})$. In the worst case, the number of candidates to be updated is $N \cdot (\text{Max}(\text{Len}(Q_i)) - \omega)$. In our incremental distance calculation, the intermediate distance for each candidate is updated only for a new data entry. Therefore, the time complexity becomes $\text{O}(\log N_{dw} + N \cdot \text{Max}(\text{Len}(Q_i)))$. We note that the actual number of candidates is much smaller than that in the worst case since the selectivity of the query sequence is less than 1.0 and the early abandoning strategy also reduces candidates significantly.

We now briefly discuss about supporting other similarity models besides L_p-norms. Dynamic Time Warping (DTW) is known to be a robust distance measure for time-series that allows similar shapes to match even if they are out of phase in the time axis [12]. Keogh [12] proposed an efficient time-series indexing method based on DTW. For exact indexing, this method uses a lower bounding function that can be used to prune false alarms without false dismissals. Our algorithm can exploit this lower bounding function directly for indexing query sequences and searching candidates that satisfy a tolerance under the DTW distance measure. To support variable lengths under DTW, we can use the subsequence matching technique proposed by Wong and Wong [24], which also uses the window construction mechanism for subsequence matching and the lower bounding function [12] under DTW. Longest Common Subsequence (LCSS) is another similarity model that provides robustness to noise. Similar to the case of DTW, we can exploit the indexing method proposed by Vlachos et al. [22] to support LCSS. Consequently, by replacing the indexing and the matching methods with those for DTW and LCSS, we can still use the main idea of this paper, i.e., the intervaled sequences and the window construction mechanism. This means that we can efficiently support variable-length and variable-tolerance under these similarity models without losing the advantages of our method. However, there still remain unsolved issues such as the candidate management and the early abandoning strategy. We leave dealing with these similarity models as a future study.

6. Performance evaluation

In this section, we explain the results of performance evaluation. We describe the experimental data and environments in Section 6.1 and present results of the experiments in Section 6.2.

6.1. Experimental data and environments

We compare performance of our algorithm with those of existing methods: the naive approach (let us call it *Naive*), Gao-1 proposed by Gao and Wang [9], and MSM Proposed by Lian et al. [14]. For fair comparison, we use early abandoning techniques not only for our algorithm but also for these existing methods. Gao-2 and MSM supports fixed-length and fixed-tolerance query sequences only. Experiments show that there are not significant differences (say, less than 36%) in performance among SSM-IS, Gao-2, and MSM for fixed-length and fixed-tolerance sequences. Thus, we do not further evaluate fixed-length and fixed-tolerance ones. For variable-length and variable-tolerance query sequences, we compare our method with MSM only, which is the more recent one, modifying it to handle variable-length and variable-tolerance query sequences:

- *Naive*: This is a straightforward method that finds similar sequences by computing distances for all registered query sequences whenever a new data entry arrives.
- *Gao-1* [9]: This is a main-memory-based method that supports variable-length and variable-tolerance query sequences. The performance of Gao-1 depends on correctness of the prediction and the number of predicted data entries. In our experiments, we use the Err_{linear} prediction model for predicting 200 data entries, which leads to the best performance according to the literature [9].
- *MSM* [14]: This is a main-memory-based method that supports fixed-length and fixed-tolerance query sequences. To support variable tolerances, we use the maximum value of tolerances among the registered query sequences. To support variable lengths, in the pruning step, we compared only the fixed-length postfix of the query sequence with the data stream sequence. For fair comparison, we use the R^*-tree instead of the grid index used in the literature [14] to index query sequences.
- *SSM-IS*: This is the method we propose in this paper.

We have performed extensive experiments using two types of data sets. The first one, a real stock data set,[4] consists of 8000 entries. We call this data set *STOCK-DATA*. The second one contains random walk data. The random work 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. The random work data are widely used in time-series database [8,17,18] and similar sequence matching on data streams [9,10] to simulate real time-series data such as sensor data and stock prices. We call this data set *WALK-DATA*.

We generate query sequences from the data stream sequences by taking subsequences starting from random offsets. To experiment *variable-length* query sequences, we set $\mathrm{Len}(Q)$ to a random value in $[L/2, 2 \cdot L]$ where L is a base length. We use 128, 256, 512, and 1024 as the base length L. However, for MSM, we fix $\mathrm{Len}(Q)$ as the base length L since it cannot support variable lengths. In addition, we obtain the tolerance ϵ given with the query sequence based on the selectivity. Here, the selectivity of the query sequence Q at the time point t is defined as in Eq. (6):

[4] This data set is the NASDAQ price data, which can be obtained from http://www.economagic.com/em-cgi/data.exe/sp/day-nsdc.

$$selectivity(Q,t) = \frac{\#\text{ of data sequences that are in }\epsilon\text{-match with }Q\text{ from the data stream during the time points }[1,t]}{\#\text{ of data sequences of length Len}(Q)\text{ generated from the data stream during the time points }[1,t]} \quad (6)$$

Next, to experiment *variable-tolerance* query sequences, we set the selectivity of each query sequence Q as a random value in $[s/2, 2 \cdot s]$ where s is a base selectivity. We use from 1% to 32% in exponential steps as the base selectivity s. However, for Gao-2 that cannot support variable tolerances, we make a range query with the maximum tolerance, as explained in Section 4.

All the experiments are conducted on a SUN Ultra 60 workstation with 512 Mbytes of main memory. As the multidimensional index to store query sequences, we use R^*-tree and set the page size to 4096 bytes. As lower-dimensional transformation, we use the Wavelet transformation [7] and set the number of dimensions to 6 as has been done in the literature [8,17,18].[5] We set the window size ω to the minimum length of query sequences unless mentioned otherwise.

6.2. Experimental results

6.2.1. Performance analysis

We compare the elapsed time of Naive, Gao-1, and MSM with that of SSM-IS to show the performance of similar sequence matching against variable-length and variable-tolerance query sequences. For fair comparison, SSM-IS reads the index and registered query sequences into main memory in advance since both Naive, Gao-1, and MSM are main-memory-based methods. We note that SSM-IS uses the prematching technique and its cost is included in the performance measurement. To avoid effects of noise, we experiment with 5000 data entries and use the average as the result.

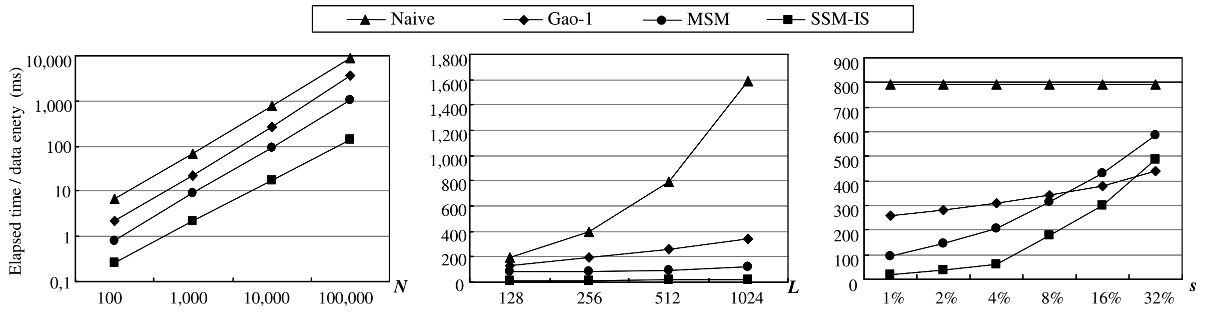
We vary the number N of registered query sequences, the base length L of the query sequence, and the base selectivity s over STOCK-DATA and WALK-DATA. We first explain in detail the results for STOCK-DATA, and then briefly mention those for WALK-DATA.

(1) STOCK-DATA: Fig. 8 shows the results of experiments for STOCK-DATA. Fig. 8(a) shows the elapsed time of each method as N is varied, illustrating scalability of each method. Fig. 8(b) and (c) shows the elapsed time as the query sequence length Len(Q) or the selectivity is varied, respectively. The results of Fig. 8 indicate that SSM-IS significantly improves the performance compared with other methods. This enhancement is due to the fact that SSM-IS computes distances only for the candidates that are found from searching index while Naive and Gao-1 compute them for all registered query sequences. MSM produces a large number candidates since the maximum value of tolerances is used to support variable tolerances. Even though MSM can efficiently prune false alarms, this large number of candidates degrades the performance.

In Fig. 8(a), SSM-IS outperforms Naive by up to 64.6 times, Gao-1 by up to 25.8 times, and MSM by up to 7.6 times. Especially, when the number of query sequences is 100,000, SSM-IS takes only 0.13 s to process similar sequence matching whenever a new data entry arrives, while MSM, Gao-1, and Naive takes 1.05, 3.56, and 8.91 s, respectively.

Fig. 8(b) indicates that the performance difference between SSM-IS and Naive or between SSM-IS and Gao-1 increases as the length of query sequence increases. The elapse time of Naive grows fastest among all three methods because the distance computation of Naive is proportional to the length of query sequences. In the case of Gao-1, the elapsed time also grows as the length of query sequences increases since the overhead of correcting the errors between the predicted entries and the corresponding actual ones increases as the length of query sequence increases. However, Gao-1 grows a lot slower than Naive because the overhead occurs only for the query sequences that are in the error range. SSM-IS is the least affected by the length of query sequences. The major reason is that SSM-IS finds candidates by searching the index regardless of the length of query sequences and only computes distances for these candidates. Moreover, by using the incremental distance calculation, SSM-IS abandons candidates early that exceed the tolerance. MSM is less affected by the

[5] A formal analysis about the number of dimensions in the Wavelet transformation is provided by Chan and Fu [4].



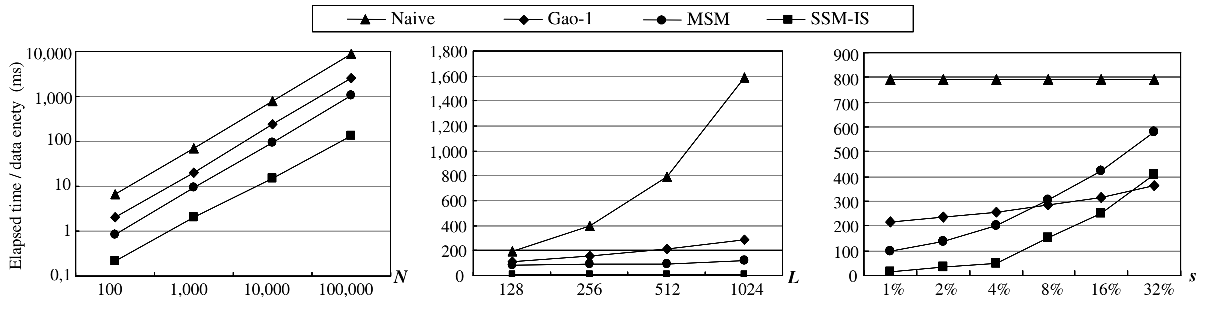
(a) Varying N ($L = 512$, $s = 1\%$) (b) Varying L ($N = 10{,}000$, $s = 1\%$) (c) Varying s ($N = 10{,}000$, $L = 512$)

Fig. 8. Comparison of the elapsed times of SSM-IS, Naive, and Gao-1 for STOCK-DATA.

length of query sequences since the method also uses the index and prunes false alarms early by the multi-scaled approximation. In summary, SSM-IS outperforms Naive by up to 102.1 times, Gao-1 by up to 22.1 times, and MSM by up to 7.7 times.

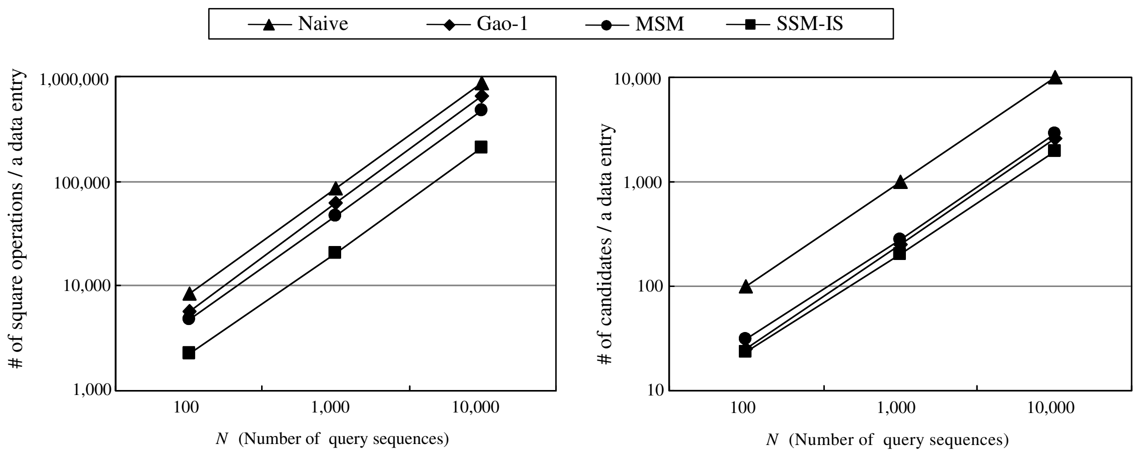
Fig. 8(c) shows that the performance improvement of SSM-IS is outstanding when the selectivity is low. The reason is that, in SSM-IS, the distance computation cost decreases as the selectivity decreases since the number of candidates retrieved by searching the index decreases. On the other hand, the elapsed time of SSM-IS increases as the selectivity increases since the number of candidates increases. In Gao-1, the elapsed time also increases as the selectivity increases since the number of query sequences whose prediction errors need to be corrected increases. However, the elapsed time of Naive does not change as the selectivity is varied since distance computation occurs for all the query sequences regardless of the selectivity. In MSM, the elapsed time rapidly increases as the selectivity increases since a very large number of candidates are retrieved from the index by using the maximum value of tolerances to support variable-tolerance query sequences (e.g., when s is 32%, almost 64% of the query sequences registered are retrieved as candidates). In summary, SSM-IS outperforms Naive by up to 47.8 times, Gao-1 by up to 15.2 times, and MSM by up to 5.5 times. However, for a high selectivity (i.e., $\geqslant$30%), the performance of SSM-IS is slightly worse than those of Gao-1 and MSM. For example, when the base selectivity s is 32%, the performance of SSM is worse than that of Gao-1 by 1.1 times. This happens because, for a high selectivity, the set of candidates retrieved by searching the index is almost identical to the entire set of query sequences, and thus, the advantage of using the index diminishes. However, we conclude that SSM-IS is much more practical in large database applications since these applications tend to select a small number of meaningful query sequences from a large database [17].

(2) WALK-DATA: Fig. 9 shows the results of experiments for WALK-DATA. As shown in the figure, the results are similar to those of STOCK-DATA in Fig. 8.



(a) Varying N ($L = 512$, $s = 1\%$) (b) Varying L ($N = 10{,}000$, $s = 1\%$) (c) Varying s ($N = 10{,}000$, $L = 512$)

Fig. 9. Comparison of the elapsed times of SSM-IS, Naive, and Gao-1 for WALK-DATA.



(a) CPU Cost (Varying N, $L = 512$, $s = 1\%$) (b) I/O Cost (Varying N, $L = 512$, $s = 1\%$)

Fig. 10. Comparison of the CPU and I/O costs of SSM-IS, Naive, Gao-1, and MSM for STOCK-DATA.

6.2.2. CPU and I/O cost analysis

We compare the CPU cost and the I/O cost of Naive, Gao-1, and MSM with those of SSM-IS. As the CPU cost, we count the number of square operations for calculating distances since this operation is the most expensive in Matching. As the I/O cost, we count the number of candidates after the pruning since the actual time-series query sequences of the candidates should be accessed from the database for refinement. In the main-memory-based experiments conducted in this paper, we consider accessing an actual query sequence as an I/O operation since this would incur an I/O access if the database were stored in disk. In the case of Naive, the number of candidates is equal to the number of query sequences. Here, we use STOCK-DATA and varies the number of query sequences N.

Fig. 10 shows the results of experiments for analyzing CPU and I/O costs. We can see that SSM-IS outperforms MSM, Gao-1, and Naive for both CPU and I/O costs. Fig. 10(a) indicates that SSM-IS needs much smaller number of square operations since SSM-IS computes distances only for the candidates that are found from searching the index while Naive and Gao-1 compute them for all the registered query sequences. In the case of MSM, the CPU cost becomes large since it uses the maximum value of tolerances. In Fig. 10(b), the number of candidates for SSM-IS and those for MSM and Gao-1 are similar. The reason is that only the query sequences that passed the step of pruning false alarms become candidates in these methods. However, we can see that Naive produces a much larger number of candidates since all the query sequences become candidates for similar sequence matching.

7. Conclusions

We proposed a new method for similar sequence matching on data streams, which we call *Similar Sequence Matching based on Intervaled Sequence* (SSM-IS). SSM-IS efficiently supports variable-length and variable-tolerance continuous query sequences for large databases stored in disk. Supporting variable lengths and variable tolerances is important in applications such as real-time sensor data, stock prices, and trajectories of moving objects.

The contributions of this paper are summarized as follows. First, to support variable-tolerance continuous query sequences, we have proposed the new notion of an *intervaled sequence*. The intervaled sequence is a sequence whose individual entries are an interval of real numbers rather than a real number itself. To support variable tolerances, we transform each query sequence to an intervaled sequence by using the tolerance as the interval. Second, to support variable-length query sequences, we have proposed using the window construction mechanism. The window construction mechanism enables us to support variable-length query sequences by dividing long sequences into smaller windows for indexing and searching the sequences. Third, we have proposed index building and matching algorithms of SSM-IS. Then, we have proved the correctness of the algorithm, i.e., the algorithm does not incur any false dismissal. Fourth, we have introduced new notions of *precandidates*, which are defined as the future candidates of similar sequences, and *prematching*, which prefetches not only current candidates but also precandidates in advance. Fifth, we have shown superiority of our algorithms through extensive experiments. When the query selectivities are low (<32%), which are practically useful cases in large database applications, experimental results show that SSM-IS outperforms the naive one by 2–102.1 times and the existing methods in the literature by 1.4–9.8 times over the entire ranges of parameters tested.

Overall, these results indicate that our method gives an excellent framework for various similar sequence matching applications for time-series data streams such as those from emerging sensor networks and ubiquitous environments.

Acknowledgement

This work was supported by the Korea Science and Engineering Foundation (KOSEF) and the Korean Government (MOST) through the NRL Program (No. R0A-2007-000-20101-0).

Appendix A

Proof of Lemma 1: We prove it by the contraposition of Lemma 1. The contraposition of Lemma 1 is "If a sequence Y is not in δ-interval match with an intervaled sequence X^{δ}, then $dist(X,Y) > \epsilon$". By Definition 3, if Y is not in δ-interval match with X^{δ}, then $Y[i] < X[i] - \epsilon$ or $Y[i] > X[i] + \epsilon$ holds for at least one i where $1 \leqslant i \leqslant \mathrm{Len}(X)$. Then, $|X[i] - Y[i]| > \epsilon$ holds for at least one i, and $dist(X,Y) > \epsilon$ holds. □

Appendix B

Proof of Corollary 1: By Theorem 1, for the data stream sequence $D[t' - \mathrm{Len}(Q) + 1 : t']$ to be ϵ-match with the query sequence Q at the time point t', the disjoint window $DW_j^{\delta}(= Q^{\delta}[\omega \cdot (j-1) + 1 : \omega \cdot j])$ should be in δ-interval match with the sliding window $SW_j(= D[(t' - \mathrm{Len}(Q) + 1) + (\omega(j-1)) : (t' - \mathrm{Len}(Q) + 1) + (\omega \cdot j - 1)])$ for at least one j where $1 \leqslant j \leqslant p$. Suppose that, as the result of searching the index, the sliding window $D[t - \omega + 1 : t]$ is in δ-interval match with the ith intervaled window DW_i^{δ} at the time point t. Then, $j = i$ holds, and $t = (t' - \mathrm{Len}(Q) + 1) + (\omega \cdot i - 1)$ also holds for the sliding window and the intervaled window DW_i^{δ} located in the corresponding position. Thus, Q is a precandidate that will be identified to be whether in ϵ-match or not at the time point $t + \mathrm{Len}(Q) - \omega \cdot i$ because $t' = t + \mathrm{Len}(Q) - \omega \cdot i$. □

References

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

[2] B. Babcock et al., Models and issues in data stream systems, in: Proceedings of the 21st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), Madison, Wisconsin, June 2002, pp. 1–16.

[3] N. Beckmann et al., The R^*-tree: an efficient and robust access method for points and rectangles, in: Proceedings of the International Conference on Management of Data, ACM SIGMOD, Atlantic City, New Jersey, May 1990, pp. 322–331.

[4] K.P. Chan, A.W.C. Fu, Efficient time series matching by wavelets, in: Proceedings of the 15th IEEE International Conference on Data Engineering (ICDE), Sydney, Australia, March 1999, pp. 126–133.

[5] S. Chandrasekaran, M.J. Franklin, Streaming queries over streaming data, in: Proceedings of the 28th International Conference on Very Large Data Bases, Hong Kong, China, August 2002, pp. 203–214.

[6] Y. Chen et al., SpADe: on shape-based pattern detection in streaming time series, in: Proceedings of the 23rd IEEE International Conference on Data Engineering (ICDE), Istanbul, Turkey, April 2007, pp. 786–795.

[7] C.K. Chui, An Introduction to Wavelets, Academic Press, 1992.

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

[9] L. Gao, X.S. Wang, Continually evaluating similarity-based pattern queries on a streaming time series, in: Proceedings of the International Conference on Management of Data, ACM SIGMOD, Madison, Wisconsin, June 2002, pp. 370–381.

[10] L. Gao, Z. Yao, X.S. Wang, Evaluating continuous nearest neighbor queries for streaming time series via pre-fetching, in: Proceedings of the International Conference on Information and Knowledge Management, ACM CIKM, McLean, Virginia, 2002, pp. 485–492.

[11] D.S. Hirschberg, Algorithms for the longest common subsequence problem, Journal of the ACM 24 (4) (1977) 664–675.

[12] E. Keogh, Exact indexing of dynamic time warping, in: Proceedings of the 28th International Conference on Very Large Data Bases, Hong Kong, China, 2002, pp. 406–417.

[13] E. Keogh et al., LB Keogh supports exact indexing of shapes under rotation invariance with arbitrary representations and distance measures, in: Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 2006, pp. 882–893.

[14] X. Lian et al., Similarity match over high speed time-series streams, in: Proceedings of the 23rd IEEE International Conference on Data Engineering (ICDE), Istanbul, Turkey, April 2007, pp. 1086–1095.

[15] H. Lim, J. Lee, M. Lee, K. Whang, I. Song, Continuous query processing in data streams using duality of data and queries, in: Proceedings of the International Conference on Management of Data, ACM SIGMOD, Chicago, Illinois, June 2006, pp. 313–324.

[16] W. Loh, S. Kim, K. Whang, A subsequence matching algorithm that supports normalization transformation in time-series databases, Data Mining and Knowledge Discovery 9 (1) (2004) 5–28.

[17] Y. Moon, K. Whang, W. Loh, Duality-based subsequence matching in time-series databases, in: Proceedings of the 17th IEEE International Conference on Data Engineering (ICDE), Heidelberg, Germany, April 2001, pp. 263–272.

[18] Y. Moon, K. Whang, W. Han, General match: a subsequence matching method in time-series databases based on generalized windows, in: Proceedings of International Conference on Management of Data, ACM SIGMOD, Madison, Wisconsin, June 2002, pp. 182–392.

[19] R. Motwani et al., Query processing, approximation, and resource management in a data stream management system, in: Proceedings of the First Biennial Conference on Innovative Data Systems Research, Asiloma, California, January 2003, pp. 245–256.

[20] Y. Sakurai, C. Faloutsos, M. Yamamuro, Stream monitoring under the time warping distance, in: Proceedings of the 23rd IEEE International Conference on Data Engineering (ICDE), Istanbul, Turkey, April 2007, pp. 1046–1055.

[21] A.F. Sheta, K. De Jong, Time-series forecasting using GA-tuned radial basis functions, Information Sciences 133 (3–4) (2001) 221–228.

[22] M. Vlachos et al., Indexing multi-dimensional time-series with support for multiple distance measures, in: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, August 2003, pp. 216–225.

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

[24] T.S.F. Wong, M.H. Wong, Efficient subsequence matching for sequences databases under time warping, in: Proceedings of the Seventh International Database Engineering and Applications Symposium (IDEAS), Hong Kong, China, July 2003, pp. 139–148.

[25] B. Yi, H.V. Jagadish, C. Faloutsos, Efficient retrieval of similar time sequences under time warping, in: Proceedings of the 14th IEEE International Conference on Data Engineering (ICDE), Orlando, Florida, February 1997, pp. 201–208.

[26] B. Yi, C. Faloutsos, Fast time sequence indexing for arbitrary L_p norms, in: Proceedings of the 26th International Conference on Very Large Data Bases, Cairo, Egypt, September 2000, pp. 385–394.

[27] M. Zhou, M.H. Wong, A segment-wise time warping method for time scaling searching, Information Sciences 173 (1–3) (2005) 227–254.