Subsequence Matching in Data Streams

Machiko Toyoda and Yasushi Sakurai

Abstract

Subsequence matching is a basic problem in the field of data stream mining, and dynamic time warping (DTW) is a powerful similarity measure often used for subsequence matching; however, the straightforward method using DTW incurs a high computation cost. In this article, we describe two subsequence matching problems in data streams—(1) the similarity between a query sequence and a data stream and (2) the similarity between data streams—and we present effective algorithms to solve these problems. We also introduce some applications using these algorithms.

1. Introduction

Data streams have attracted interest in various fields (theory, databases, data mining, and networking) because of their many important applications including financial analysis, sensor network monitoring, moving object trajectories, web click-stream analysis, and network traffic analysis. Efficient monitoring of time-series data streams is a fundamental requirement for these applications, and subsequence matching is an important technique for this. We consider two problems with subsequence matching over data streams: the similarity between a query sequence and data stream and the similarity between data streams. In the former problem, one is a fixed sequence and the other is an evolving sequence. This approach works well if we have already determined the patterns we want to find. By contrast, the latter problem focuses on co-evolving sequences and reveals hidden patterns between them without preliminary knowledge.

Unlike the traditional setting, the sampling rates of streams are often different, and their time period varies in practical situations. Subsequence matching should address asynchronous data and should be robust against noise and provide scaling along the time axis. We focus on dynamic time warping (DTW) as a similarity measure for subsequence matching over data streams. DTW is typically applied to limited situations in an offline manner. Since data streams arrive online at high bit rates and are potentially unbounded in size, the computation time and memory space increase greatly. Ideally, we need a solution that can return correct results without any omissions, even at high speed.

This article presents the streaming algorithms SPRING and CrossMatch for subsequence matching. These are one-pass algorithms that are strictly based on DTW and that guarantee correct results without any omissions. We describe the basic ideas behind each algorithm and show how these algorithms work in relation to data stream processing. Moreover, we focus on sensor network monitoring as an application of subsequence matching and discuss the differences between the two algorithms.

2. Background

2.1 Related work

Similarity search over time-series data has been studied for many years. Most methods focus on similarity queries for static datasets with a query sequence. Specifically, given a query sequence and a distance threshold, a similarity query finds all the sequences or subsequences that are similar to the query sequence. The basic idea is to transform the sequence into the frequency domain using a discrete Fourier transform (DFT) and then to extract the few features from the resultant frequency-domain sequence [1]. Euclidean distance is used as a similarity measure. Other feature

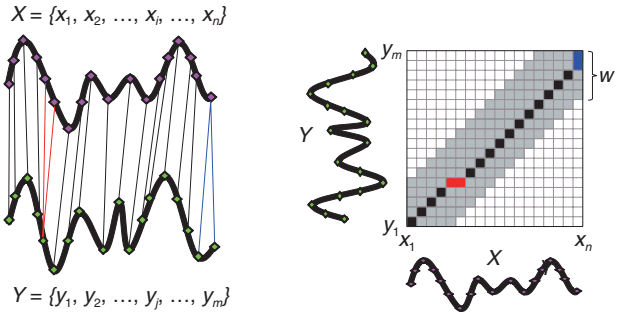

Fig. 1. Illustration of DTW.

extraction functions include discrete wavelet transform (DWT) [2], singular value decomposition (SVD) [3], piecewise aggregate approximation (PAA) [4], and adaptive piecewise constant approximation (APCA) [5]. These functions have been proposed to reduce the number of dimensions of the time-series.

Since the Euclidean distance treats sequence elements independently, it cannot be used to calculate the distance between sequences whose lengths are different. DTW has been adopted to overcome these problems [6]; it is a widely used similarity measure that allows time-axis scaling. The DTW distance can be computed using dynamic programming techniques. Therefore, DTW is typically a much costlier approach than the Euclidean distance. To address the cost issue, several methods have been proposed that use a lower bound to refine the results and envelope techniques to constrain the computation cells [7]–[10].

Similarity search over data streams has recently attracted more research interest. In contrast to a setting where the data sequences are fixed, data streams arrive irregularly and are frequently updated. Therefore, incremental computation techniques that use the previous feature when computing the new feature are required. Several discriminative methods have been proposed for subsequence matching with a query sequence. These include subsequence matching based on prediction [11], approximate subsequence matching with data segmentation and piecewise line representation [12], subsequence matching supporting shifting and scaling in the time and amplitude domains [13], and subsequence matching based on DTW with batch filtering [14].

Another important area is pattern discovery between data streams. Mueen et al. [15] presented the first online motif discovery algorithm to accurately monitor and maintain motifs, which represent repeated subsequences in time-series, in real time. Papadimitriou et al. [16] proposed an algorithm for discovering optimal local patterns, which concisely describe the main trends in data streams.

In summary, there have been many previous studies on similarity search over data streams; however, none of them have addressed effective subsequence matching based on DTW. We formally define two problems for subsequence matching and present efficient and effective algorithms to solve them.

2.2 DTW

DTW is a transformation that allows sequences to be stretched along the time axis to minimize the distance between them. The DTW distance of two sequences is the sum of the tick-to-tick distances after the two sequences have been optimally warped to match each other. An illustration of DTW is shown in **Fig. 1**. The left figure is the alignment by DTW for measuring the DTW distance. To align two sequences, we construct a time warping matrix as shown in the right figure (the warping scope w will be described in section 4.1). The warping path is a set of grid cells in the time warping matrix; this set of grid cells represents the optimal alignment between the sequences. Consider two sequences, $X = (x_1, x_2, ..., x_n)$ of length n and $Y = (y_1, y_2, ..., y_m)$ of length m. Their DTW distance $D(X, Y)$ is defined as

$$D(X, Y) = d(n, m)$$
$$d(i, j) = \|x_i - y_j\| + min \begin{cases} d(i, j-1) \\ d(i-1, j) \\ d(i-1, j-1) \end{cases} \quad (1)$$

$$d(0, 0) = 0, d(i, 0) = d(0, j) = \infty$$

$$(i = 1, ..., n; j = 1, ..., m;).$$

Note that $\|x_i - y_j\| = (x_i - y_j)^2$ is the distance between two numerical values in cell (i, j) of the time warping matrix. Note that other choices (e.g., absolute difference $\|x_i - y_j\| = |x_i - y_j|$) can also be used; the algorithms we present in this article are completely independent of the choice made. Specifically, DTW requires $O(nm)$ time because the time warping matrix consists of nm elements. Note that the space complexity is $O(m)$ because the algorithm needs only two columns (i.e., the current and previous columns) of the time warping matrix to compute the DTW distance.

In the problem of subsequence matching with a

query sequence, given an evolving sequence $X = (x_1, x_2, ..., x_n)$ and a fixed-length query sequence $Y = (y_1, y_2, ..., y_m)$, we want to find the subsequences of X that are similar to Y in the sense of the DTW distance. By contrast, in the problem of subsequence matching between data streams, given two evolving sequences $X = (x_1, x_2, ..., x_n)$ and $Y = (y_1, y_2, ..., y_m)$, we want to find the subsequence pairs, namely the common local patterns over data streams. We will give the exact definitions for these problems later (in sections 3.1 and 4.1). In both settings, naïve ways of subsequence matching require unfeasible computation time and memory consumption in data stream processing. We show that our approaches offer considerable improvement without loss of accuracy.

3. Subsequence matching with a query sequence

3.1 Problem definition

Data stream X is a discrete, semi-infinite sequence of numbers $x_1, x_2, \ldots, x_n, \ldots$, where x_n is the most recent value. Note that n increases with every new time-tick. Let $X[i_s : i_e]$ be the subsequence of X that starts from time-tick i_s and ends at i_e. The subsequence matching problem is to find the subsequence $X[i_s : i_e]$ that is highly similar to a fixed-length query sequence Y (i.e., the subsequence with a small value of $D(X[i_s : i_e], Y)$. However, a subtle point should be noted: whenever the query Y matches a subsequence of X (e.g., $X[i_s : i_e]$), we expect that there will be several other matches with subsequences that heavily overlap the *local minimum* best match. Overlaps provide the user with redundant information and would slow down the algorithm since all useless solutions are tracked and reported. In our solution, we discard all these extra matches. Specifically, the problem we want to solve is as follows:

Problem 1 Given a stream X (that is, an evolving data sequence, which at the time of interest has length n), a query sequence Y of fixed-length m, and a distance threshold ε, report all subsequences $X[i_s : i_e]$ such that

1. the subsequences are close enough to the query sequence: $D(X[i_s : i_e], Y) \leq \varepsilon$, and
2. among several overlapping matches, report only the local minimum; that is, $D(X[i_s : i_e], Y)$ is the smallest value from the set of overlapping subsequences that satisfy the first condition.

Hereafter we use the term *optimal subsequences* to refer to subsequences that satisfy both conditions.

3.2 SPRING

The most straightforward (and slowest) solution to this problem is to consider all possible subsequences $X[i_s : i_e]$ $(1 \leq i_s \leq i_e \leq n)$ and apply the standard DTW dynamic programming algorithm, which requires $O(n^2)$ matrices. The time complexity would be $O(n^3m)$ (or $O(n^2m)$ per time-tick). This method is extremely expensive. Moreover, it cannot be extended to the streaming case. To solve the problem, we therefore proposed SPRING [17], which is an efficient algorithm. SPRING uses only a single matrix and detects optimal subsequences in stream processing.

3.2.1 Basic ideas

Our solution to the problem is based on two ideas. The first idea is *star-padding*, in which sequence Y is prefixed with a special value ("*") that always gives zero distance. This value stands for the *don't care* interval, that is, the interval $(-\infty : +\infty)$.

Definition 1 (Star-padding) Given a query sequence Y, star-padding of Y is defined as follows.

$$Y' = (y_0, y_1, y_2, ..., y_m)$$
$$y_0 = (+\infty : -\infty) \tag{2}$$

We use Y' to compute the DTW distances of Y and subsequences of X, instead of operating on the original sequence of Y.

Given a sequence $Y = (y_1, y_2, ..., y_m)$, we have the star-padding of Y. Let X be a sequence of length n. We can then derive the minimum distance $D(X[i_s : i_e], Y)$ from the matrix of X and Y'.

$$D(X[i_s : i_e], Y) = d(i_e, m) = min(d(i, m))$$

$$d(i, j) = \|x_i - y_j\| + d_{best}$$

$$d_{best} = min \begin{cases} d(i, j-1) \\ d(i-1, j) \\ d(i-1, j-1) \end{cases} \tag{3}$$

$$d(i, 0) = 0, d(0, j) = \infty$$

$$(i = 1, ..., n; j = 1, ..., m).$$

Star-padding dramatically reduces both time and space since we need to update only $O(m)$ numbers per time-tick to derive the minimum distance.

Star-padding is a good first step, and it can tell us (a) where the subsequence match ends and (b) what

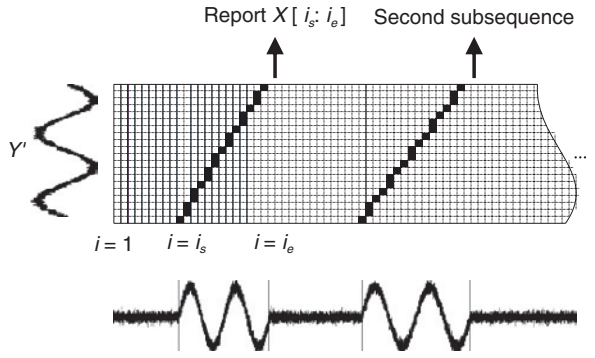

Fig. 2. Illustration of SPRING algorithm, which uses only a single matrix to capture all qualifying subsequences.

the resulting distance is. However, such applications often also need the starting time-tick of the match. Our second idea is to use a *subsequence time warping matrix* (STWM): we augment the time warping matrix and record the starting position of each candidate subsequence.

Definition 2 (STWM) The STWM contains the value $d(i, j)$, which is the best distance to match the prefix of length i from X with the prefix of length j from Y, and the starting position $s(i, j)$ corresponding to $d(i, j)$. The starting position is computed as

$$s(i,j) = \begin{cases} s(i,j-1) & (d(i,j-1) = d_{best}) \\ s(i-1,j) & (d(i-1,j) = d_{best}) \\ s(i-1,j-1) & (d(i-1,j-1) = d_{best}) \end{cases} \quad (4)$$

We obtain the starting position of $D(X[i_s : i_e], Y)$ as

$$i_s = s(i_e, m). \quad (5)$$

We update the starting position accompanied by the distance value as well as the distance value itself. By using the matrix, we can identify which subsequence gave the minimum distance during stream processing.

3.2.2 Algorithm

The way SPRING detects optimal subsequences is shown in **Fig. 2**. SPRING is carefully designed to (a) guarantee no false dismissals for the second condition of Problem 1 and (b) report each match as early as possible (detailed proofs are given in [17]). For each incoming data point x_n, we first incrementally update the m distance values $d(n, j)$ and determine the m starting positions $s(n, j)$. The algorithm reports the subsequence after confirming that the current optimal subsequence cannot be replaced by the upcoming subsequences. That is, we report the subsequence that gives the minimum distance d_{min} when the $d(n, j)$ and $s(n, j)$ arrays satisfy

$$\forall j,\ d(n,j) \geq d_{min}\ V\ s(n,j) > i_e, \quad (6)$$

which means that the captured optimal subsequence cannot be replaced by the upcoming subsequences. Otherwise, the upcoming candidate subsequences would not overlap the captured optimal subsequence. We initialize d_{min} and the $d(n, j)$ arrays after the output.

Example 1 Assume that $\varepsilon = 15$, $X = (5, 12, 6, 10, 6, 5, 13)$, and $Y = (11, 6, 9, 4)$ in **Fig. 3**. The cell (i, j) of the matrix contains $d(i, j)$ and $s(i, j)$. At $i = 3$, we found candidate subsequence $X[2 : 3]$ whose distance $d(3, 4) = 14$ below ε. At $i = 4$, although the distance $d(4, 4) = 38$ is larger than ε, we do not report $X[2 : 3]$ since $d(4, 3) = 2$, which means $X[2 : 3]$ can be replaced by the upcoming subsequences. We then capture the optimal subsequence $X[2 : 5]$ at $i = 5$. $X[2 : 5]$ is reported at $i = 7$ since we now know that none of the upcoming subsequences will be the optimal subsequence. Finally, because subsequences starting from $i = 7$ may be candidates for the next group, we do not initialize $d(7, 1)$.

4. Subsequence matching between data streams

4.1 Problem definition

We have focused on finding subsequences similar to a query sequence. In this section, we address subsequence matching between data streams. That is, both sequences X and Y are co-evolving data streams, and we want to identify common local patterns between them.

Like data stream X, data stream Y is a discrete, semi-infinite sequence of numbers $y_1, y_2, \ldots, y_m, \ldots$, where y_m is the most recent value. Note that m increases with every new time-tick. Let $Y[j_s : j_e]$ be the subsequence of Y that starts from time-tick j_s and ends at j_e. The lengths of $X[i_s : i_e]$ and $Y[j_s : j_e]$ are $l_x = i_e - i_s + 1$ and $l_y = j_e - j_s + 1$, respectively. The goal is to find the common local patterns of sequences by data stream processing based on DTW. That is, we want to detect subsequence pairs that satisfy

$$D(X[i_s : i_e], Y[j_s : j_e]) \leq \varepsilon L(l_x, l_y), \quad (7)$$

$y_4 = 4$		54 (1)	110 (2)	14 (2)	38 (2)	6 (2)	7 (2)	88 (2)
$y_3 = 9$		53 (1)	46 (2)	10 (2)	2 (2)	10 (4)	17 (4)	18 (4)
$y_2 = 6$		37 (1)	37 (2)	1 (2)	17 (4)	1 (4)	2 (4)	51 (4)
$y_1 = 11$		36 (1)	1 (2)	25 (3)	1 (4)	25 (5)	36 (6)	4 (7)
x_i		5	12	6	10	6	5	13
i		1	2	3	4	5	6	7

Fig. 3. Illustration of SPRING. The upper numbers show the distance in each element of the matrix. The numbers in parentheses show the starting position.

where L is a function that sets the length of the subsequence. The DTW distance increases as the subsequence length increases since it is the sum of the distances between elements. Therefore, unlike the problem where sequence Y is fixed, the distance threshold should be proportional to the subsequence length to detect the subsequence pairs without depending on the subsequence length. Accordingly, we set the distance threshold at $\varepsilon L(l_x, l_y)$. In this article, the algorithm uses $L(l_x, l_y) = (l_x + l_y)/2$, which is the average length of the two subsequences, but the user can make any other choice (e.g., $L(l_x, l_y) = max(l_x, l_y)$ or $L(l_x, l_y) = min(l_x, l_y)$).

Equation (7) allows us to detect subsequence pairs without regard to the subsequence length. In practice, however, we might detect shorter and meaningless matching pairs owing to the influence of noise. We introduce the concept of subsequence match length to enable us to discard such meaningless pairs. We formally define the *cross-similarity* between X and Y, which indicates common local patterns.

Definition 3 (Cross-similarity) Given two sequences X and Y, a distance threshold ε, and a threshold of subsequence length l_{min}, $X[i_s : i_e]$ and $Y[j_s : j_e]$ have the property of cross-similarity if this sequence pair satisfies the condition

$$D(X[i_s : i_e], Y[j_s : j_e]) \leq \varepsilon (L(l_x, l_y) - l_{min}). \quad (8)$$

The minimum length l_{min} of subsequence matches should be given by the users. The subsequences that satisfy this equation are guaranteed to have lengths exceeding l_{min}.

In addition to subsequence matching with a query sequence, we need to consider several other matches that strongly overlap the local minimum best match. Specifically, in this setting, an overlap is simply where two subsequence pairs have a common alignment, which is defined as follows:

Definition 4 (Overlap) Given two warping paths for subsequence pairs of X and Y, their overlap is defined as the condition where the paths share at least one element.

Our solution for subsequence matching between data streams is to detect the local best subsequences from the set of overlapping subsequences. Thus, our goal is to find the best match of cross-similarity.

Problem 2 Given two sequences X and Y, thresholds ε, and l_{min}, report all subsequence pairs $X[i_s : i_e]$ and $Y[j_s : j_e]$ that satisfy the following conditions.

1. $X[i_s : i_e]$ and $Y[j_s : j_e]$ have the property of cross-similarity.
2. $D(X[i_s : i_e], Y[j_s : j_e]) - \varepsilon (L(l_x, l_y) - l_{min})$ is the minimum value among the set of overlapping subsequence pairs that satisfies the first condition.

Hereafter we use the term *qualifying* subsequence pairs to refer to pairs that satisfy the first condition, and we use *optimal* subsequence pairs to refer to pairs that satisfy both conditions.

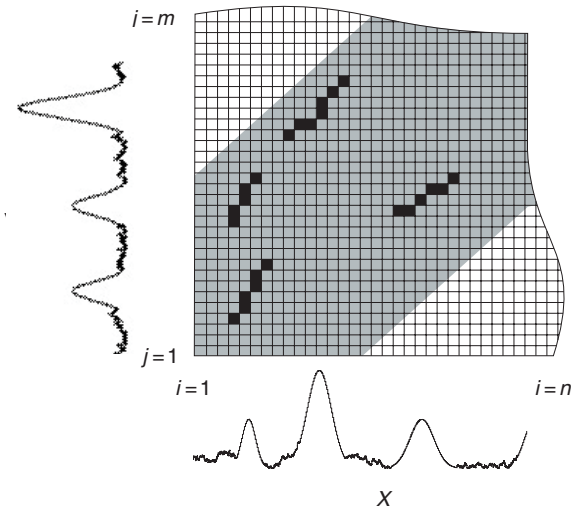

Fig. 4. Illustration of CrossMatch. Black cells indicate the warping paths of the optimal subsequence pairs, and gray cells indicate the warping scope.

Typically, new elements in data streams are usually more significant than those in the distant past. To limit the number of cells in the matrix and focus on recent elements, we utilize a global constraint for DTW, namely the Sakoe-Chiba band [18] that restricts the warping path to the $|i - j| \le w$ range (i.e., the gray cells in Fig. 1), where w is called the warping scope. In data stream processing, we compute the cells from a recent element to an earlier element of the warping scope w. If $m = n$, the warping scope is exactly equal to the Sakoe-Chiba band.

4.2 CrossMatch

To solve the problem of cross-similarity, the most straightforward solution is to consider all possible subsequences of $X[i_s : i_e]$ $(1 \le i_s < i_e \le n)$, and all possible subsequences of $Y[j_s : j_e]$ $(1 \le j_s < j_e \le m)$ in the warping scope and apply the standard DTW dynamic programming algorithm. Let w be the warping scope (the gray cells in the figure). The solution requires $O(nw^2+mw^2)$ time (per update) and space because it has to handle a total of $O(nw+mw)$ matrices to compute the DTW distance. If we prune dissimilar subsequence pairs and reduce the number of matrices, the distance computations become much more efficient. Our method, CrossMatch [19], finds good matches in a single matrix efficiently by pruning the subsequences (see **Fig. 4**).

4.2.1 Basic ideas

To identify the dissimilar subsequences early, we propose computing the DTW distance indirectly by using a *scoring function*. The scoring function computes the maximum cumulative score corresponding to the DTW distance with a score matrix. The score is determined by accumulating the difference between the threshold and the distance between the elements in the score matrix. Thus, we can recognize a dissimilar subsequence pair since the score has a negative value if the subsequence pair does not satisfy the first condition of Problem 2. The scoring function initializes the negative score to zero and then restarts the computation from the cell. This operation allows us to discard unqualifying, non-optimal subsequence pairs.

Definition 5 (Score matrix) Given two sequences, $X = (x_1, x_2, ..., x_n)$ and $Y = (y_1, y_2, ..., y_m)$, and the warping scope w, score $V(X[i_s : i_e], Y[j_s : j_e])$ of $X[i_s : i_e]$ and $Y[j_s : j_e]$ defined as follows:

$$V(X[i_s : i_e], Y[j_s : j_e]) = v(i_e, j_e)$$

$$v(i, j) = max \begin{cases} 0 \\ \varepsilon b_v - \|x_i - y_j\| + v(i, j-1) \\ \varepsilon b_h - \|x_i - y_j\| + v(i-1, j) \\ \varepsilon b_d - \|x_i - y_j\| + v(i-1, j-1) \end{cases} \quad (9)$$

$$v(0, 0) = v(i, 0) = v(0, j) = 0$$
$$(i = 1, ..., n;\ j = 1, ..., m;$$
$$n - w \le i \le n;\ m - w \le j \le m;).$$

Symbols b_v, b_h, and b_d in Eq. (9) indicate a weight function for each direction, which makes transformation between the score and the DTW distance possible. These values are determined as subsequence length L. For example, for $L(l_x, l_y) = (l_x+l_y)/2$, we obtain $b_v = b_h = 1/2$ and $b_d = 1$, respectively. The scoring function is designed so that the sum of the weights on the warping path is equal to subsequence length L. Therefore, it guarantees transformation between the DTW distance and the score, and it finds the qualifying subsequence pairs without any omissions (detailed proofs are given in [19]). The DTW distance of a subsequence pair is computed from the score and the subsequence length as

$$D(X[i_s:i_e], Y[j_s:j_e]) = \varepsilon L(l_x, l_y) - V(X[i_s:i_e], Y[j_s:j_e]) \quad (10)$$
$$\text{s.t. } V(X[i_s:i_e], Y[j_s:j_e]) > 0.$$

6	13			21	0	0	0
5	9		16	22	25	20	0
4	2	11	0	0	27	49	0
3	4	13	0	0	36	42	0
2	9	0	11	26	24	0	
1	11	0	13	19	1		
j	y_j \ x_i	5	12	10	6	3	18
	i	1	2	3	4	5	6

(a) Score matrix

6	13			(1,3)	(4,6)	(5,6)	(6,6)
5	9		(1,3)	(1,3)	(2,1)	(2,1)	(6,5)
4	2	(1,3)	(2,4)	(3,4)	(2,1)	(2,1)	(6,4)
3	4	(1,3)	(2,3)	(3,3)	(2,1)	(2,1)	(6,3)
2	9	(1,2)	(2,1)	(2,1)	(2,1)	(5,2)	
1	11	(1,1)	(2,1)	(2,1)	(2,1)		
j	y_j \ x_i	5	12	10	6	3	18
	i	1	2	3	4	5	6

(b) Position matrix

Fig. 5. Example of cross-similarity detection. The lightly shaded cells signify cross-similarity, and the dark cell in each matrix shows the best match.

Equation (10) holds for the time warping and the score matrices, which have the same starting position (i_s, j_e).

The scoring function tells us the subsequence match ends and the subsequence score. However, we lose the information about the starting position of the subsequence. This is the motivation behind our second idea, a position matrix: we store the starting position to keep track of qualifying subsequence pairs in a streaming fashion.

Definition 6 (Position matrix) The position matrix stores the starting position of each subsequence pair. The starting position $p(i, j)$ corresponding to score $v(i, j)$ is computed as:

$$p(i,j) = \begin{cases} p(i, j-1) & (v(i, j-1) \neq 0 \ \Lambda \ v(i,j) \\ & \quad = \varepsilon b_v - \|x_i - y_j\| + v(i, j-1)) \\ p(i-1, j) & (v(i, j-1) \neq 0 \ \Lambda \ v(i,j) \\ & \quad = \varepsilon b_h - \|x_i - y_j\| + v(i-1, j)) \\ p(i-1, j-1) & (v(i-1, j-1) \neq 0 \ \Lambda \ v(i,j) \\ & \quad = \varepsilon b_d - \|x_i - y_j\| + v(i-1, j-1)) \\ (i, j) & (otherwise) \end{cases} \quad (11)$$

The starting position is described as a coordinate value; $p(i_e, j_e)$ indicates the starting position (i_s, j_s) of the subsequence pair $X\,[i_s : i_e]$ and $Y\,[j_s : j_e]$. We can identify the optimal subsequence that gives the maximum score during stream processing by using the score and position matrices. Moreover, the starting position of the shared cell is maintained through the subsequent alignments because we repeat the operation, which maintains the starting position of the selected previous cell. Thus, we know the overlapping subsequence pairs from the fact that the starting positions match.

4.2.2 Algorithm

We now have all the pieces needed to answer the question: how do we find the optimal subsequence pairs? Every time x_n or y_m is received at time-tick n or m, our CrossMatch algorithm incrementally updates the score and starting position and retains the end position. We use a candidate array to find the optimal subsequence pair, and we store the best pair in a set of overlapping subsequence pairs. CrossMatch reports the optimal subsequence pair after confirming that it cannot be replaced by the upcoming subsequence pairs. The upcoming candidate subsequence pairs do not overlap the captured optimal subsequence pair if the starting positions in the position matrix satisfy the condition:

$$(\forall i, p(i, m) = C_s) \ \Lambda \ (\forall j, p(n, j) \neq C_s). \quad (12)$$

CrossMatch requires only $O(w)$ (i.e., constant) time (per update) and space. This is a great reduction because the straightforward solution requires $O(nw^2+mw^2)$ time (per update) and space.

Example 2 Assume that we have two sequences X = (5, 12, 6, 10, 3, 18) and Y = (11, 9, 4, 2, 9, 13) as well as $\varepsilon = 14$, $l_{min} = 2$, and $w = 3$ in **Fig. 5**. To simplify our example with no loss of generality, we assume that x_i

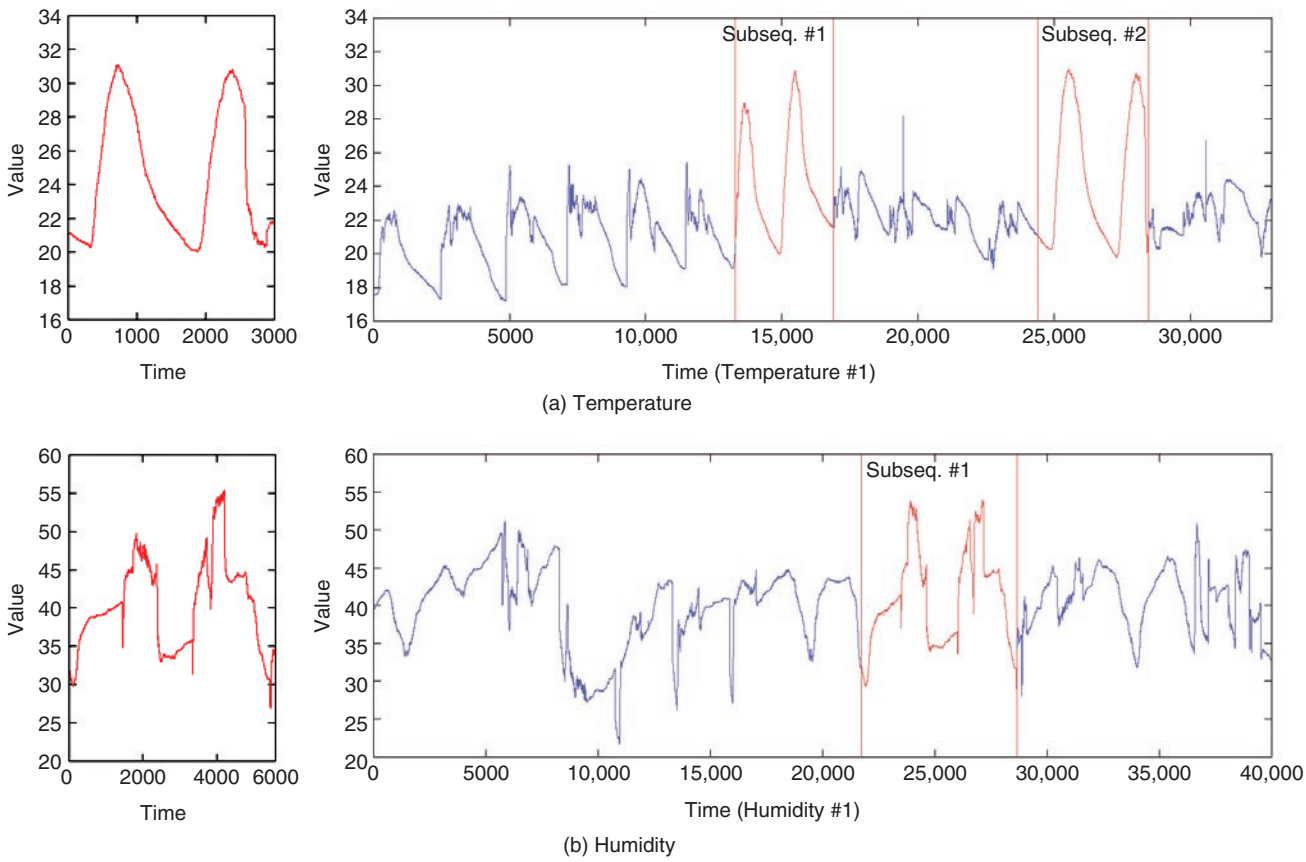

Fig. 6. Discovery of similar subsequences using SPRING. The left and right columns show the query and data sequences, respectively.

and y_j arrive alternately. At each time-tick, the algorithm updates the scores and the starting positions. At $i = 4$, we update the cells from (4, 1) to (4, 3) and identify a candidate subsequence, $X[2 : 4]$ and $Y[1 : 3]$, starting at (2, 1), whose score $v(4, 3) = 36$ is greater than εl_{min}. At $j = 4$, we update the cells from (1, 4) to (4, 4). Although we detect no subsequences satisfying the condition, we do not report the subsequence $X[2 : 4]$ & $Y[1 : 3]$ since this pair might be replaced by upcoming subsequences. We then capture the optimal subsequence pair $X[2 : 5]$ & $Y[1 : 4]$ at $i = 5$. Finally, we report the subsequence as the optimal subsequence at $j = 6$ since we can confirm that none of the upcoming subsequences can be optimal.

5. Applications of SPRING and CrossMatch

One example of subsequence matching application is sensor network monitoring. In sensor networks, sensors send their readings frequently. Each sensor produces a stream of data, and these streams need to be monitored and combined to detect changes in the environment that may be of interest to users. Users are likely to be interested in one or more sensors within a particular spatial region. These interests are expressed as trends and similar patterns.

The optimal subsequences that SPRING detected in temperature and humidity datasets are shown in **Fig. 6**. The problem that SPRING solves is very simple. Users assign the pattern for which they wish to search. The detected subsequences may be normal patterns, abnormal patterns, frequent patterns, or trends depending on the query sequences.

By contrast, CrossMatch focuses on the commonality between data sequences and reveals hidden local patterns. The left graphs in **Fig. 7** show similar subsequence pairs in each dataset (i.e., the left and center graphs) as the optimal warping paths. These pairs can be frequent patterns or trends and may be query

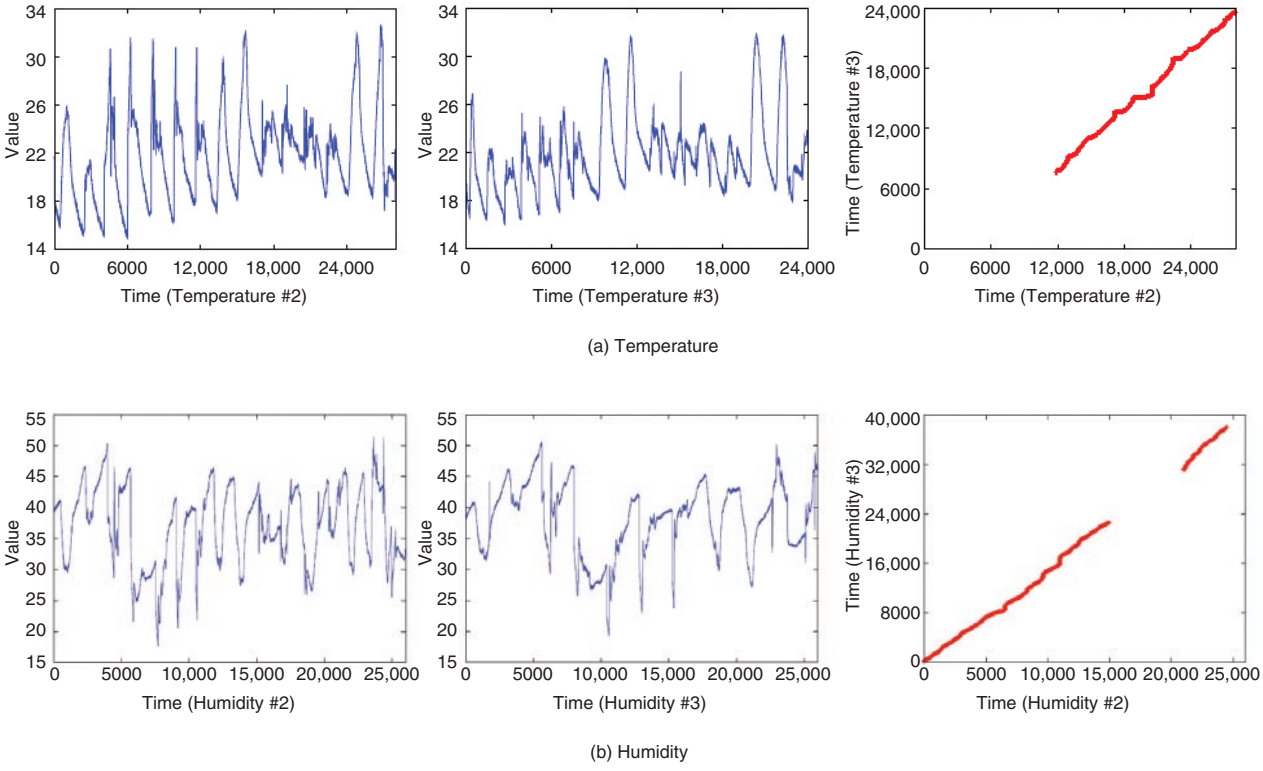

Fig. 7. Discovery of cross-similarity by CrossMatch. The left and center graphs show data sequences, and the right graphs show the warping paths for the optimal subsequence pairs.

sequences that users want to search for in the future. This focuses on the *similarities* of datasets. If we look at the *dissimilarities*, we can see different applications. For example, the result in Fig. 7(b) can be interpreted as showing that two sensors behaving in the same way have dissimilar intervals. In this case, we can assume a sensor failure and an anomaly in the environment.

6. Conclusion

In this article, we summarized subsequence matching and presented two one-pass algorithms, SPRING and CrossMatch, for subsequence matching over data streams. Both algorithms process at high-speed, exhibit constant time and space consumption, and guarantee correct results. Subsequence matching is applied in several domains, and DTW is a powerful similarity measure for subsequence matching. We believe that our algorithms will contribute to the development of many different intelligent applications.

References

[1] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos, "Fast Subsequence Matching in Time-series Databases," Proc. of ACM SIGMOD'94 International Conference on Management of Data, pp. 419–429, Minneapolis, MN, USA, 1994.
[2] I. Popivanov and R. J. Miller, "Similarity search over time-series data using wavelets," Proc. of IEEE 18th International Conference on Data Engineering (ICDE), pp. 212–224, San Jose, CA, USA, 2002.
[3] K. V. Ravi Kanth, D. Agrawal, A. E. Abbadi, and A. Singh, "Dimensionality Reduction for Similarity Searching in Dynamic Databases," Proc. of ACM SIGMOD'98 International Conference on Management of Data, pp. 166–176, Seattle, WA, USA, 1998.
[4] B. -K. Yi and C. Faloutsos, "Fast Time Sequence Indexing for Arbitrary Lp Norms," Proc. of the 26th International Conference on Very Large Databases (VLDB), pp. 385–394, Cairo, Egypt, 2000.
[5] E. J. Keogh, K. Chakrabarti, S. Mehrotra, and M. -J. Pazzani, "Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases," Proc. of ACM SIGMOD'01 International Conference on Management of Data, pp. 151–162, Santa Barbara, CA, USA, 2001.
[6] D. J. Berndt and J. Clifford, "Finding Patterns in Time Series: a Dynamic Programming Approach," Advances in Knowledge Discovery and Data Mining, pp. 229–248, 1996.
[7] I. Assent, M. Wichterich, R. Krieger, H. Kremer, and T. Seidl, "Anticipatory DTW for Efficient Similarity Search in Time Series Databases," Proc. of the VLDB Endowment (PVLDB), Vol. 2, No. 1, pp. 826–837, 2009.
[8] E. J. Keogh, "Exact Indexing of Dynamic Time Warping," Proc. of the 28th International Conference on Very Large Data Bases (VLDB), pp.

406–417, Hong Kong, China, 2002.
[9] Y. Sakurai, M. Yoshikawa, and C. Faloutsos, "FTW: Fast Similarity Search under the Time Warping Distance," Proc. of the 24th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS), pp. 326–337, Baltimore, MD, USA, 2005.
[10] Y. Zhu and D. Shasha, "Warping Indexes with Envelope Transforms for Query by Humming," Proc. of ACM SIGMOD'03 International Conference on Management of Data, pp. 181–192, San Diego, CA, USA, 2003.
[11] X. Lian and L. Chen, "Efficient Similarity Search over Future Stream Time Series," IEEE Trans. on Knowledge and Data Engineering (TKDE), Vol. 20, No. 1, pp. 40–54, 2008.
[12] H. Wu, B. Salzberg, and D. Zhang, "Online Event-driven Subsequence Matching over Financial Data Streams," Proc. of ACM SIGMOD'04 International Conference on Management of Data, pp. 23–34, 2004.
[13] Y. Chen, M. A. Nascimento, B. C. Ooi, and A. K. H. Tung, "SpADe: On Shape-based Pattern Detection in Streaming Time Series," Proc. of IEEE 23rd International Conference on Data Engineering (ICDE), pp. 786–795, Istanbul, Turkey, 2007.
[14] M. Zhou and M. H. Wong, "Efficient Online Subsequence Searching in Data Streams under Dynamic Time Warping Distance," Proc. of IEEE 24th International Conference on Data Engineering (ICDE), pp. 686–695, Cancun, Mexico, 2008.
[15] A. Mueen and E. Keogh, "Online Discovery and Maintenance of Time Series Motifs," Proc. of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), pp. 1089–1098, Washington D.C., USA, 2010.
[16] S. Papadimitriou and P. Yu, "Optimal Multiscale Patterns in Time Series Streams," Proc. of ACM SIGMOD'06 International Conference on Management of Data, pp. 647–658, Chicago, IL, USA, 2006.
[17] Y. Sakurai, C. Faloutsos, and M. Yamamuro, "Stream Monitoring under the Time Warping Distance," Proc. of IEEE 23rd International Conference on Data Engineering (ICDE), pp. 1046–1055, Istanbul, Turkey, 2007.
[18] H. Sakoe and S. Chiba, "Dynamic Programming Algorithm Optimization for Spoken Word Recognition," IEEE Trans. on Acoustics, Speech and Signal Processing, Vol. 26, No. 1, pp. 43–49, 1978.
[19] M. Toyoda, Y. Sakurai, and Y. Ishikawa, "Pattern Discovery in Data Streams under the Time Warping Distance," VLDB Journal, 2012.

Machiko Toyoda
Researcher, NTT Communication Science Laboratories.

She received the B.S. and M.S. degrees from Ochanomizu University, Tokyo, in 2004 and 2006, respectively, and the Ph.D. degree in information science from Nagoya University, Aichi, in 2012. She joined NTT Information Sharing Platform Laboratories in 2006 and moved to NTT Communication Science Laboratories in 2010. She has been engaged in research on data stream mining. Her research interests include similarity search and time-series analysis. She received the Computer System Symposium Best Poster Award in 2006 and the 1st Forum on Data Engineering and Information Management (DEIM) Best Paper Award in 2009. She is a member of the Institute of Electronics, Information and Communication Engineers (IEICE), the Information Processing Society of Japan (IPSJ), and the Database Society of Japan (DBSJ).

Yasushi Sakurai
Senior Research Scientist, NTT Communication Science Laboratories.

He received the B.E. degree from Doshisha University, Kyoto, in 1991, and the M.E. and Ph.D. degrees in engineering from Nara Institute of Science and Technology in 1996 and 1999, respectively. He joined NTT Cyber Space Laboratories in 1998. He was a visiting researcher at Carnegie Mellon University during 2004–2005. His research interests include indexing, data mining, and data stream processing. He received the IPSJ Nagao Special Researcher Award in 2007, the DBSJ Kambayashi Incentive Award (Young Scientist Award) in 2007, and 12 best paper awards, including two KDD Best Research Paper Awards in 2008 and 2010, IPSJ Best Paper Awards in 2004 and 2008, and an IEICE Best Paper Award in 2008. He is a member of the Association for Computing Machinery, IEICE, IPSJ, and DBSJ.