Discovery of Cross-Similarity in Data Streams

Machiko Toyoda [†], Yasushi Sakurai [‡]
[†]*NTT Information Sharing Platform laboratories*
[†]*Graduate School of Information Science, Nagoya University*
[†]toyoda.machiko@lab.ntt.co.jp
[‡]*NTT Communication Science laboratories*
[‡]yasushi.sakurai@acm.org

Abstract— **In this paper, we focus on the problem of finding partial similarity between data streams. Our solution relies on dynamic time warping (DTW) as a similarity measure, which computes the distance between sequences whose lengths and/or sampling rates are different. Instead of straightforwardly using DTW that requires a high computation cost, we propose a streaming method that efficiently detects partial similarity between sequences. Our experiments demonstrate that our method detects pairs of optimal subsequences correctly and that it significantly reduces resources in terms of time and space.**

I. Introduction

The challenge of processing and analyzing data streams is now a major topic in several communities (e.g., financial data analysis [1], sensor network monitoring [2], moving object search [3], web click-stream analysis [4], and network traffic analysis [5]). Many applications require detecting hidden patterns that may exist in co-evolving data streams and subsequence matching is one of important technique. We assume two similarities for subsequence matching over data streams. One is the similarity between a query sequence and data stream and the other is the similarity between data streams. Existing techniques on online subsequence matching mainly focus on the former [6][7]. In contrast, we address the partial similarity between data streams for frequent pattern discovery, rule mining, and outlier detection. Unlike the traditional settings in which the length of query is fixed, the problem is more challenging since the starting position and the length of the subsequence changes dynamically. Additionally, since the sampling rates of streams are frequently different and their time period varies in practical situation, a similarity measure for subsequence matching should be robust against misalignments between the sequences. We use the dynamic time warping (DTW) distance to solve the problem. DTW is a robust and widely used measure, and is suitable for subsequence matching since it provides time scaling (such as stretching or shrinking a portion of a sequence along the time axis).

In this paper, we present an efficient method to automatically detect all similar subsequence pairs between data streams. Our method works continuously in a streaming fashion and is based on DTW. Our experiments demonstrate that our method correctly detects the optimal subsequence pairs and improves the performance and the memory space as we expected.

II. Problem Definition

A. Background – Dynamic Time Warping

Intuitively, a DTW distance of two sequences is the sum of tick-to-tick distances after two sequences have been optimally warped to match each other. The alignment between sequences is called for the warping path and is represented as a set of cells give the DTW distance. Let us formally 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:

$$\begin{aligned}
&D(X,Y) = d(n,m) \\
&d(i,j) = ||x_i - y_j|| + min \begin{cases} d(i,j-1) \\ d(i-1,j) \\ d(i-1,j-1) \end{cases} \\
&d(0,0) = 0, \quad d(i,0) = d(0,j) = \infty \\
&(i = 1,...,n;\ j = 1,...,m),
\end{aligned} \tag{1}$$

where $||x_i - y_j|| = (x_i - y_j)^2$ is the distance between two numerical values. Notice that any other choice would be fine; our algorithms are completely independent of such choices. Specifically, DTW requires $O(mn)$ time since the time warping matrix consists of mn elements. Note that the space complexity is $O(m)$ per time-tick since the algorithm needs only two columns (i.e., the current and previous columns) of the time warping matrix to compute the DTW distance.

B. Cross-similarity

A 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 starting from time-tick i_s and ending at i_e, and let $Y[j_s : j_e]$ be the subsequence of Y starting from time-tick j_s and ending 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. Our goal is to find partial similarity between multiple sequences, based on DTW, in data stream processing. More concretely, we want to detect subsequence pairs that satisfy

$$D(X[i_s : i_e], Y[j_s : j_e]) \leq \varepsilon L(l_x, l_y), \tag{2}$$

where $D(X[i_s : i_e], Y[j_s : j_e])$ is the DTW distance between $X[i_s : i_e]$ and $Y[j_s : j_e]$, and L is a function that provides the length of the subsequence pair. In this paper, we use $L(l_x, l_y) = (l_x + l_y)/2$, the average length of the two subsequences, but we can use any other choice (e.g., $L(l_x, l_y) = max(l_x, l_y)$ or $L(l_x, l_y) = min(l_x, l_y)$). The DTW distance increases as the subsequence length grows since it is the sum of distances between elements. Therefore, the distance threshold should be proportional to the subsequence length, thus we set it to $\varepsilon L(l_x, l_y)$.

We formally define the '*cross-similarity*' between X and Y.

Definition 1 (Cross-similarity): Given two sequences X and Y, a distance threshold ε, and a threshold of subsequence length l_{min}, cross-similarity of the subsequences $X[i_s : i_e]$ and $Y[j_s : j_e]$ satisfies:

$$D(X[i_s : i_e], Y[j_s : j_e]) \leq \varepsilon(L(l_x, l_y) - l_{min}). \qquad (3)$$

The minimum length l_{min} of subsequence matches should be given by users. To satisfy cross-similarity, the length L of the subsequence pair should be greater than l_{min} since the DTW distance gives the value greater than zero. That is, cross-similarity includes the concept of subsequence length and we detect subsequence pairs whose lengths are longer than l_{min}. Without this concept, the algorithm might detect shorter and meaningless matching pairs. We discard them and detect the optimal pairs that satisfy 'real' user requirements.

Additionally, we should mention the following point: whenever subsequences of X and Y ($X[i_s : i_e]$ and $Y[j_s : j_e]$) match, there will be several other matches by subsequences, which heavily overlap with the 'local minimum' best match. An overlap means that the warping paths of subsequence pairs cross and corresponds to the following case. The warping paths, which have the common starting position, separate on the way. These matches would be doubly detrimental: (a) they could potentially confuse the user with redundant information and (b) they would slow down the algorithm to keep track of and report all the useless 'solutions'. We detect the local best subsequence pairs among a set of overlapping subsequence pairs. Thus, the main issue we want to accomplish is to find the best match of cross-similarity.

Problem 1: Given two sequences X and Y, thresholds ε and l_{min}, we want to find subsequence pairs, $X[i_s : i_e]$ and $Y[j_s : j_e]$, satisfying 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 in each group of overlapping subsequence pairs that satisfy the first condition.

Our previous algorithm [8] to find pairs of similar subsequences over data streams uses an original similarity measure and outputs all overlapping subsequences. Our upcoming algorithm gets rid of them and provides the best results.

We use a 'qualifying' subsequence pair hereafter, to specifically denote the subsequence pair that satisfies the first condition, and use an 'optimal' subsequence pair to specifically denote the subsequence pair that satisfies both of conditions.

III. Discovery of cross-similarity

A. Naive solution

For this problem, the most straightforward solution would be to consider all possible subsequences of $X[i_s : i_e]$ $(1 \leq i_s < i_e \leq n)$ and all possible subsequences of $Y[j_s : j_e]$ $(1 \leq j_s < j_e \leq m)$ and apply the standard DTW dynamic programming algorithm. We refer to this method as *Naive*.

Let $d_{i,j}(p, q)$ be the distance of the element (p, q) in the time warping matrix that starts from i on the x-axis and j on the y-axis. The distance of the subsequence matching between X and Y can be obtained as follows.

$$\begin{aligned}
&D(X[i_s : i_e], Y[j_s : j_e]) = d_{i_s,j_s}(l_x, l_y)\\
&d_{i,j}(p,q) = ||x_{i+p-1} - y_{j+q-1}|| + d_{best}\\
&d_{best} = \begin{cases} d_{i,j}(p, q-1)\\ d_{i,j}(p-1, q)\\ d_{i,j}(p-1, q-1)\end{cases}\\
&d_{i,j}(0,0) = 0, \quad d_{i,j}(p,0) = d_{i,j}(0,q) = \infty\\
&(i = 1, ..., n;\ p = 1, ..., n-i+1;\\
&\qquad j = 1, ..., m;\ q = 1, ..., m-j+1)
\end{aligned} \qquad (4)$$

The processing of the naive solution updates the distance arrays of incoming x_i at time-tick i and that of incoming y_j at time-tick j about all possible subsequences, and then determines the one in which $D(X[i_s : i_e], Y[j_s : j_e]) - \varepsilon(L(l_x, l_y) - l_{min})$ is the minimum value in each overlapping group.

The naive solution updates $O(m + n)$ DTW distances for incremental computation on each time warping matrix when an element of the sequence arrives since we need only two columns, i.e., the current and previous columns, per single matrix. The naive solution requires $O(mn)$ matrices to compute the DTW distance because it has to create a new matrix for every time-tick. Therefore, the naive solution requires $O(m^2n + mn^2)$ time and space for incremental computation.

B. Proposed solution

Our solution is based on the following ideas.

1) Scoring function: To compute the distance of every possible subsequence pair, the naive solution creates a new time warping matrix for every time-tick. Instead of using the DTW function, we compute the DTW distance indirectly using a scoring function. The scoring function is essentially based on a dynamic programming approach, and optimally warps given sequences to match each other. Our function differs in that we compute the *maximum cumulative score* corresponding to the DTW distance.

Given two sequences $X = (x_1, ..., x_i, ..., x_n)$ and $Y = (y_1, ..., y_j, ..., y_m)$, we can then derive the score $V(X[i_s : i_e], Y[j_s : j_e])$ of $X[i_s : i_e]$ and $Y[j_s : j_e]$ as follows.

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

$$v(i,j) = max \begin{cases} 0 \\ w_v\varepsilon + v(i, j-1) - ||x_i - y_j|| \\ w_h\varepsilon + v(i-1, j) - ||x_i - y_j|| \\ w_d\varepsilon + v(i-1, j-1) - ||x_i - y_j|| \end{cases} \quad (5)$$

$$v(0,0) = v(i,0) = v(0,j) = 0.$$

To keep track of the optimal subsequence pair, we reset the score and restart the score computation from the current element if the cumulative score is a negative value. This means Definition 1 is no longer satisfied for the subsequence pair of X and Y ending at (i, j). The scoring function identifies the qualifying subsequence pairs with a single matrix and updates only $O(m + n)$ scores for incremental computation.

The symbols w_v, w_h, and w_d in Eq. (5) indicate the weight of each direction, determined by L. For example, for $L(l_x, l_y)$, the current L value increases by $1/2$ if the score of a vertical or horizontal element is inherited, and it increases by 1 if the score of a diagonal element is inherited. Thus, we obtain $w_v = w_h = 1/2$ and $w_d = 1$. The scoring function is designed so that the sum of the weight on the warping path (i.e., w_v, w_h, and w_d) is equal to $L(l_x, l_y)$. With this property, we can easily transform the score into the DTW distance. Problem 1 for the scoring function is equivalent to the following conditions:

1) $V(X[i_s : i_e], Y[j_s : j_e]) \geq \varepsilon l_{min}$
2) $V(X[i_s : i_e], Y[j_s : j_e]) - \varepsilon l_{min}$ is the maximum value in each group of subsequence pairs that warping path crosses.

2) Position matrix: The scoring function tells us the subsequence match ends and the score. However, we lose the information about the starting position of the subsequence pair. We introduce a position matrix. The element (i, j) of the position matrix contains the starting position $s(i, j)$ $(i = 1, ..., n;\ j = 1, ..., m)$. The starting position $s(i, j)$ is computed as follows:

$$s(i,j) = \begin{cases} s(i, j-1) & (v(i,j-1) \neq 0 \ \wedge \ v(i,j) \\ & \quad = w_v\varepsilon + v(i,j-1) - ||x_i - y_j||) \\ s(i-1, j) & (v(i-1,j) \neq 0 \ \wedge \ v(i,j) \\ & \quad = w_h\varepsilon + v(i-1,j) - ||x_i - y_j||) \\ s(i-1, j-1) & (v(i-1,j-1) \neq 0 \ \wedge \ v(i,j) \\ & \quad = w_d\varepsilon + v(i-1,j-1) - ||x_i - y_j||) \\ (i,j) & (otherwise). \end{cases} \quad (6)$$

We choose the same element in the position matrix and inherit the starting position stored in the element if we choose a vertical, horizontal, or diagonal element. We choose (i, j) as the starting position in the position matrix if all scores of neighboring elements are zero, or the score $v(i, j)$ is zero in the score matrix. The score computation and the update of the starting position keep exactly the same warping path. Thus, we can obtain the starting position of the optimal subsequence pair, whose score is the maximum value in a streaming fashion.

IV. Experiments

We performed experiments on synthetic datasets to evaluate the effectiveness of our method. Our experiments were conducted on a 2-GHz Intel Xeon E5335 with 4 GB of memory, running Linux.

A. Detecting cross-similarity

We present case studies for discovering the optimal subsequences i.e., the best match of cross-similarity, of CrosMatch. We set l_{min} to 500 and ε to 1.0e-4 for the *RandomSines* ,and set l_{min} to 15% of the sequence length and ε to 5.0e-6 for the *Spikes*. In Fig. 1, the left and center figures represent the datasets, and the right figure represents the optimal warping paths of cross-similarity detected from these datasets.

1) RandomSines: RandomSines consists of discontinuous sine waves with white noise (see Fig. 1 (a)) and includes different-length intervals between sine waves. The sine waves were generated using a random walk function. We varied the period of each sine wave in the sequence. The intervals between these sine waves are also different. As shown in the right figure of Fig. 1 (a), our method perfectly identifies all sine waves and time-varying periodicity. In this figure, the difference in the period of each sine wave appears as the difference in the slope.

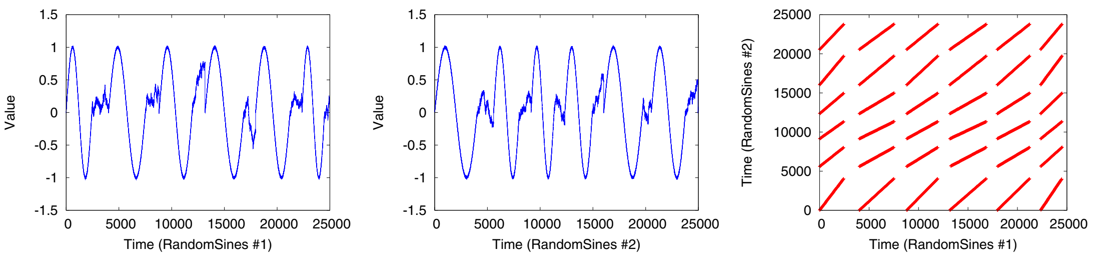
2) Spikes: Spikes consists of large and small spikes. The data of different-length intervals between spikes were generated using a random walk function. The period of each spike is also different. As seen in the right figure of Fig. 1 (b), we confirm that our method detects large spikes and small spikes. The difference in the period of each spike appears as the difference in plot length; wide spikes indicate long plot lengths and narrow spikes indicate short plot lengths.

B. Performance

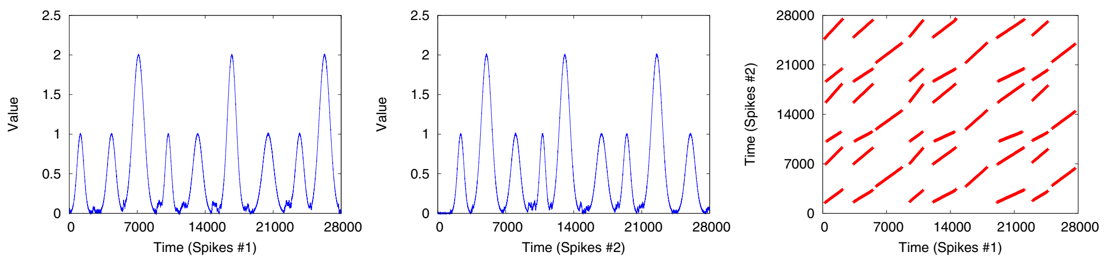
We compared our method with the naive solution, and SPRING to evaluate the efficiency and to verify the complexity of our method. SPRING is an algorithm based on DTW for finding similar subsequences to a fixed-length query sequence from data streams [7]. SPRING updates $O(m^2)$ values if we receive x_i at time-tick i, and $O(mn)$ values if we receive y_j at time-tick j to identify optimal subsequence pairs. Therefore, an algorithm with SPRING for finding cross-similarity requires $O(m^2+mn)$ time and space for incremental computation since SPRING requires $O(m)$ matrices.

Our method, the naive and the SPRING implementations are compared in terms of computation time and memory space for varying sequence lengths. We used *RandomSines* for this experiment.

Fig. 2 shows the experimental results for computation time. Time is the average processing time to compute the score and the starting position in each matrix against each sequence length. As we expected, our method identifies the optimal subsequence pairs much faster than naive and SPRING implementations. We can confirm that our method significantly reduces computation time.



(a) *RandomSines*



(b) *Spikes*

Fig. 1. Discovery of cross-similarity using *RandomSines* and *Spikes*.

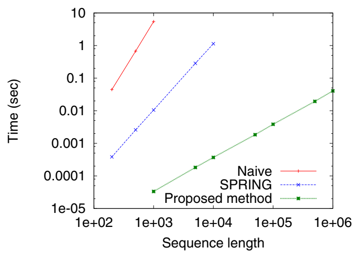


Fig. 2. Wall clock time for cross-similarity as a function of a sequence length.

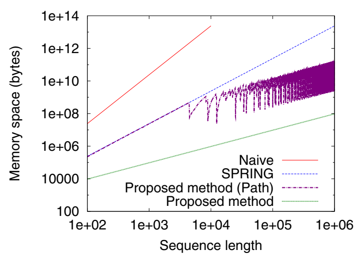


Fig. 3. Memory space consumption for cross-similarity as a function of a sequence length.

Fig. 3 compares each matrix of Naive, SPRING, and our method in terms of memory space. We performed two measurements for evaluating our method: (a) the algorithm keeps the distance and the position of the optimal subsequence pair and (b) it provides information about the warping path of the subsequence pair. The abscissa axis in Fig. 3 represents two data stream lengths. The space requirement of our method depends on the existence of the cross-similarity and increases when the algorithm keeps track of the optimal warping path. However, our method is clearly lower than that of the Naive and SPRING implementations, as shown Fig. 3.

V. Conclusions

We introduced the problem of cross-similarity and proposed a streaming method to address this problem. Our method is based on DTW and detects similar subsequence pairs between data streams. Our experiments demonstrated that our method works as expected, detecting the optimal subsequence pairs effectively and efficiently.

References

[1] H. Wu, B. Salzberg, and D. Zhang, "Online event-driven subsequence matching over financial data streams," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2004)*, June 2004, pp. 23–34.

[2] Y. Zhu and D. Shasha, "Efficient elastic burst detection in data streams," in *Proceedings of the ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2003)*, August 2003, pp. 336–345.

[3] L. C. 0002, M. T. Özsu, and V. Oria, "Robust and fast similarity search for moving object trajectories," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2005)*, June 2005, pp. 491–502.

[4] O. Nasraoui, C. Rojas, and C. Cardona, "A framework for mining evolving trends in web data streams using dynamic learning and retrospective validation," *Computer Networks*, vol. 50, no. 10, pp. 1488–1512, July 2006.

[5] F. Korn, S. Muthukrishnan, and Y. Wu, "Modeling skew in data streams," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2006)*, June 2006, pp. 181–192.

[6] M. Zhou and M. H. Wong, "Efficient online subsequence searching in data streams under dynamic time warping distance," in *Proceedings of the IEEE 24th International Conference on Data Engineering (ICDE 2008)*, April 2008, pp. 686–695.

[7] Y. Sakurai, C. Faloutsos, and M. Yamamuro, "Stream monitoring under the time warping distance," in *Proceedings of the IEEE 23th International Conference on Data Engineering (ICDE 2007)*, April 2007, pp. 1046–1055.

[8] M. Toyoda, Y. Sakurai, and T. Ichikawa, "Identifying similar subsequences in data streams," in *Proceedings of the 19th International Conference on Database and Expert Systems Applications (DEXA2008)*, ser. Lecture Notes in Computer Science, vol. 5181, September 2008, pp. 210–224.