Stream Monitoring under the Time Warping Distance

Yasushi Sakurai
NTT Cyber Space Laboratories
sakurai.yasushi@lab.ntt.co.jp

Christos Faloutsos*
Carnegie Mellon University
christos@cs.cmu.edu

Masashi Yamamuro
NTT Cyber Space Laboratories
yamamuro.masashi@lab.ntt.co.jp

Abstract

The goal of this paper is to monitor numerical streams, and to find subsequences that are similar to a given query sequence, under the DTW (Dynamic Time Warping) distance. Applications include word spotting, sensor pattern matching, and monitoring of bio-medical signals (e.g., EKG, ECG), and monitoring of environmental (seismic and volcanic) signals. DTW is a very popular distance measure, permitting accelerations and decelerations, and it has been studied for finite, stored sequence sets. However, in many applications such as network analysis and sensor monitoring, massive amounts of data arrive continuously and it is infeasible to save all the historical data.

We propose SPRING, a novel algorithm that can solve the problem. We provide a theoretical analysis and prove that SPRING does not sacrifice accuracy, while it requires constant space and time *per time-tick. These are dramatic improvements over the naive method. Our experiments on real and realistic data illustrate that SPRING does indeed detect the qualifying subsequences correctly and that it can offer dramatic improvements in speed over the naive implementation.*

1. Introduction

Data streams have attracted the interest of various communities (theory, database, data mining, and networking), due to their many important applications, such as financial analysis, network monitoring, mobile services, and sensor network management. The most fundamental support needed in these applications is efficient monitoring of time-series data streams. Since the data streams arrive online at high bit rates and are potentially unbounded in size, the resource limitations unavoidably imply a trade-off – it is practically impossible to keep all historical data in the allotted memory, but fast query processing must be ensured.

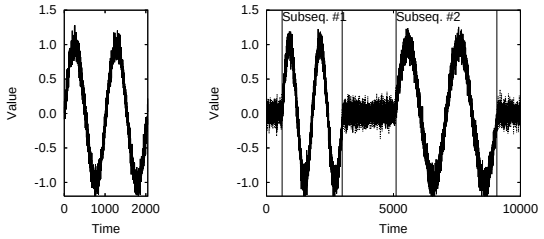

Figure 1. Illustration of stream monitoring under the DTW distance. The left and right columns show the query sequence and stream, respectively.

These applications require a subsequence-matching mechanism to monitor data streams. And in addition, since the sampling rates of streams are frequently different and their time period varies in practical situations, the mechanism should be robust against noise and provide scaling of the time axis. In this paper, we address the problem of efficiently monitoring multiple numerical streams under the DTW (Dynamic Time Warping) distance. DTW is one of the most useful distance measures because of its characteristics: DTW is a transformation that allows sequences to be stretched along the time axis to minimize the distance between the sequences.

Many algorithms have been proposed to monitor data streams in an online fashion. However, to the best of our knowledge, this is the first study that investigates time warping for monitoring data streams. Intuitively, this problem is equivalent to subsequence matching in an online fashion.

The problem is illustrated in Figure 1. The query sequence is the sinusoid pattern at the left. The stream, shown on the right, consists of three flat and noisy parts and two (noisy) sinusoids, not of the same period. Our system is able to spot the sinusoids after some stretching or shrinking. The matches are marked with vertical lines. Our system not only works continuously, in a streaming fashion, but also has dramatically better performance than a straightforward implementation in terms of speed and memory. The performance of our system does not depend on the past length of the data stream. The savings can reach and exceed several orders of magnitude.

*This material is based upon work supported by the National Science Foundation under Grants No. SENSOR-0329549 EF-0331657IIS-0326322 IIS-0534205. This work is also supported in part by the Pennsylvania Infrastructure Technology Alliance (PITA), Intel, NTT, and Hewlett-Packard. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, or other funding parties.

Our contributions are as follows:

1. We present SPRING, a new, streaming method for subsequence matching in data streams. The method is fast, accurate, and nimble, requiring constant space and time per time-tick.
2. We carefully define the problem of *disjoint queries*, a cross between best-match and range queries, so that it is suitable for a streaming setting.
3. We carried out extensive experiments on real and realistic data, which show that SPRING works as expected; SPRING is up to 650,000 times faster than the naive method.

The remainder of the paper is organized as follows. Section 2 describes related work on data streams and DTW. In Section 3 we formally define the problem of monitoring data streams under the DTW distance. We then describe SPRING, our method for solving this problem. Section 4 discusses the accuracy and complexity of SPRING. Section 5 reviews the results of the experiments, which clearly show the effectiveness of SPRING. Section 6 is a brief conclusion.

2. Related Work

Related work falls broadly into two categories. The first category includes work on DTW, where methods have been developed for sequence matching, but the focus is typically on indexing of stored data sets, not on stream processing. The other category includes work on data streams. These methods focus on comparing streams under various L^p distances, on clustering, and on summarizing. We review each category.

2.1. Sequence indexing for DTW

The Dynamic Time Warping distance (DTW) is a very popular distance function that allows for scaling along the time axis [5, 7, 11, 12]. Many sequence-matching methods for DTW have been proposed, especially in speech recognition [15] and bioinformatics [11].

Within the database community, several indexing methods for DTW have been proposed, but the focus is mainly on whole sequence matching. Yi et al. and Kim et al. [19, 9] have proposed lower bounding measures for DTW that guarantee no false dismissals. Keogh [8] proposed a search method based on global constraints that appear in dynamic programming. Global constraints (e.g., the Sakoe-Chiba Band and the Itakura Parallelogram [15]) limit the scope of the warping path. Zhu et al.'s search method [21] is also based on global constraints and represents an improvement over the one proposed by Keogh [8]. Sakurai et al. [17] proposed the FTW method with successive approximations, refinements and additional optimizations, to accelerate "whole sequence" matching under DTW.

Wong et al. studied subsequence matching for DTW [18]. They introduce a sliding window approach and propose indexing all possible prefixes with a spatial access method. Clearly, their focus is on stored data sets, as opposed to data streams.

2.2. Pattern discovery in data streams

Although none of the streaming methods deals with DTW, we review them here because they examine related topics, such as pattern discovery, summarization, and lossy compression for data streams

An interesting method using *sketches* to discover *representative trends* in time-series was proposed by Indyk et al. [4]. A representative trend is the section of a sequence with the smallest sum of "distances" among *all* other sections of the same length. This method uses random projections [6] for dimensionality reduction and FFT to quickly compute the sum of distances. Gilbert et al. [2] use wavelets to compress the data into a fixed amount of memory, by keeping track of the largest Haar wavelet coefficients and carefully updating them on-line. Guha et al. [3] solve the k-median problem in a single pass over for data streams. Zhu et al. [21] studied burst detection in streams. AWSOM [13] is one of the first streaming methods for forecasting and is intended to discover arbitrary periodicities in single time sequences.

Multiple streams have also attracted significant interest. Ganti et al. [1] proposed a generic framework for streaming mining. Zhu et al. [20] focused on monitoring multiple streams in real time. Their proposed StatStream computes the pairwise correlations among all streams. SPIRIT [14] addressed the problem of capturing correlations and finding hidden variables corresponding to trends in collections of data streams. Sakurai et al. [16] proposed BRAID, which efficiently detects lag correlations between data streams.

However, none of the above methods examines subsequence matching on streams, under the DTW distance.

3. Proposed Method

First, we define the problems and some fundamental concepts, then we describe the intuition behind our approach, and finally we give algorithms for solving the problems.

3.1. Preliminaries

3.1.1 Dynamic time warping

Intuitively, the Dynamic Time Warping (DTW) distance of two sequences is the sum of tick-to-tick distances after the two sequences have been optimally warped to match each other. Let us formally consider the two sequences $X = (x_1, x_2, \ldots, x_n)$ of length n and $Y=(y_1, y_2, \ldots, y_m)$ of length m. Their DTW distance $D(X, Y)$ is defined as:

Table 1. Symbols and definitions

Symbol	Definition
X	Data sequence/stream of length n
x_t	Value/element of X at time $t = 1, \ldots, n$
$X[t_s : t_e]$	Subsequence of X, including elements in positions t_s through t_e
Y	Query sequence of length m
y_i	i-th element of Y
Y'	Star-padding of Y
$D(X, Y)$	DTW distance between X and Y
$f_t(k, i)$	Distance of the element (k, i) in the t-th time warping matrix of X and Y
$d(t, i)$, d_i	Distance of (t, i) in the matrix of X and Y'
$s(t, i)$, s_i	Starting position of (t, i)

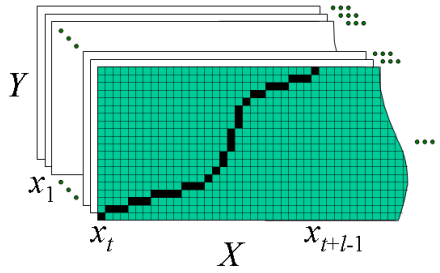

Figure 2. Illustration of subsequence matching under the DTW distance. The black squares denote the optimal warping path in the time warping matrix. The naive solution has to maintain the matrices starting from every time-tick.

$$D(X, Y) = f(n, m)$$
$$f(t, i) = \|x_t - y_i\| + \min \begin{cases} f(t, i-1) \\ f(t-1, i) \\ f(t-1, i-1) \end{cases} \quad (1)$$
$$f(0, 0) = 0, \quad f(t, 0) = f(0, i) = \infty$$
$$(t = 1, \ldots, n; i = 1, \ldots, m)$$

where $\|x_t - y_i\| = (x_t - y_i)^2$ is the distance between two numerical values. Notice that any other choice (say, absolute difference: $\|x_t - y_i\| = |x_t - y_i|$) would be fine; our algorithms are completely independent of such choices. The DTW distance is computed with the "*time warping matrix*", which stores the values of the function of Equation 1. Specifically, DTW requires $O(nm)$ time since the time warping matrix consists of nm elements. Note that the space complexity is $O(m)$ since the algorithm needs only two columns (i.e., the current and previous columns) of the time warping matrix to compute the DTW distance.

Given an evolving sequence $X(= x_1, \ldots, x_n)$ and a fixed-length query sequence $Y(= y_1, \ldots, y_m)$, we want to find the subsequences of X that are similar to Y in the sense of the DTW distance. We will give the exact definitions for this problem later (Section 3.1.2). A naive way of subsequence matching would be to compute the time warping matrices starting from every time-tick (See Figure 2). The naive solution requires $O(n^2 m)$ time to find the qualifying subsequence of length l starting from x_t (i.e., ending at x_{t+l-1}). We show that this approach can be considerably improved without loss of accuracy.

3.1.2 Problem definition

A data stream X is a discrete, semi-infinite sequence of numbers x_1, x_2, ..., x_n, ..., where x_n is the most recent value. Notice that n increases with every new time-tick. Let $X[t_s : t_e]$ denote the subsequence starting from time-tick t_s and ending at t_e. We want to find the subsequence $X[t_s : t_e]$ that has high similarity to a fixed-length query sequence Y (i.e., the subsequence with a small value of $D(X[t_s : t_e], Y)$).

When X is a fixed-length sequence, the problem can be formulated as the usual two versions: "*best-match query*", and "*range query*". The *best-match* version, with fixed-length X, is an important stepping stone. Specifically, the sub-problem we want to solve is as follows:

Problem 1 (Best-match query) *Given sequences X of length n and Y of length m, find the subsequence $X[t_s : t_e]$ whose DTW distance from Y is the smallest among those of all possible subsequences $X[t : j]$, that is, $D(X[t_s : t_e], Y) \leq D(X[t : j], Y)$ for any pair of $t = 1, \ldots, n$ and $j = t, \ldots, n$.*

The full problem we want to solve is one in which the data sequence X is actually a stream of semi-infinite length. In this case, the *best-match query* makes little sense, since we can never be sure if the future will bring up a better match than the one we have already found. The *range query* version is suitable in the streaming case. However, a subtle point should be noted: whenever the query Y matches a subsequence of X (say $X[t_s : t_e]$), we expect that there will be several other matches by subsequences which heavily overlap with the "local minimum" best match. Thus, in the standard *range query* version, we propose adding a second condition that aims to discard all these extra matches. These matches would be doubly harmful: (a) they could potentially flood the user with redundant information and (b) they would slow down the algorithm by forcing it to keep track of and report all these useless "solutions".

We shall use the term "optimal" subsequence hereafter, to denote exactly the subsequence that is the local best, among a set of overlapping, qualifying subsequences of X. Thus, the main problem we propose and solve in this work is as follows:

Problem 2 (Disjoint query) *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 threshold ϵ, report all subsequences $X[t_s : t_e]$ such that*

1. *the subsequences are close enough to the query sequence: $D(X[t_s : t_e], Y) \leq \epsilon$, and*

2. *among several overlapping matches, report only the local minimum; that is, $D(X[t_s : t_e], Y)$ is the smallest value in the group of overlapping subsequences that satisfy the first condition.*

The additional challenge is to find a streaming solution, which, at time n, will process a new value of X and report each match as early as possible.

To simplify our presentation, we will focus on the *best-match query* first, and then discuss how to handle the *disjoint queries* for data streams. Our basic ideas can be applied to both types of query.

3.1.3 Naive solution

For the *best-match* problem for a fixed length data sequence X (Problem 1), the most straightforward (and slowest) solution would be to consider all the possible subsequences $X[t_s : t_e]$ ($1 \le t_s \le t_e \le n$) and apply the standard DTW dynamic programming algorithm, which requires $O(n^2)$ matrices. The time complexity would be $O(n^3 m)$ (or $O(n^2 m)$ per time-tick). Not only is this method extremely expensive, but it also cannot be extended to the streaming case. We refer to it as *Super-Naive*.

A better solution, but still not good enough, is as follows: to find a qualifying subsequence $X[t_s : t_e]$, we would compute the distance between Y and all possible subsequences of X using $O(n)$ matrices and then choose the minimum distance. We refer to this method as *Naive*.

Let $f_t(k, i)$ be the distance of the element (k, i) in the t-th time warping matrix, which starts from t. The minimum distance of the subsequence matching between X and Y can be obtained as follows:

$$D(X[t_s : t_e], Y) = f_{t_s}(t_e - t_s + 1, m) = \min(f_t(k, m))$$
$$f_t(k, i) = \|x_{t+k-1} - y_i\| + \min \begin{cases} f_t(k, i-1) \\ f_t(k-1, i) \\ f_t(k-1, i-1) \end{cases} \quad (2)$$
$$f_t(0, 0) = 0, \;\; f_t(k, 0) = f_t(0, i) = \infty$$
$$(t = 1, \ldots, n; k = 1, \ldots, n - t + 1; i = 1, \ldots, m).$$

Since the naive solution needs $O(n)$ matrices, $O(nm)$ numbers have to be updated for each time-tick.

The processing of disjoint queries has to be done in the same way. The naive solution computes the distances of all possible subsequences, and then chooses the one that gives the minimum distance from each group of overlapping subsequences.

3.2. Basic ideas

Our solution is based on the two ideas described below.

3.2.1 Star-padding

The naive solution creates a new time warping matrix for every time-tick. Instead of the naive solution that needs $O(n)$ matrices, we propose using only a single matrix to obtain the minimum distance of subsequences of X.

Our first proposed idea is to prefix the sequence Y with a special value ("*"), that always gives zero distance. This value stands for the "don't care" interval, that is, the interval $(-\infty : +\infty)$. Let $Y = (y_1, y_2, \ldots, y_m)$ be a query sequence. We introduce its augmented version Y':

$$Y' = (y_0, y_1, y_2, \ldots, y_m) \quad (3)$$
$$y_0 = (-\infty : +\infty).$$

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

Observation 1 *Once we introduce the* star-padding, *we need only a single time-warping matrix to find the best subsequence of X.*

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, instead of $O(nm)$, which the naive solution requires. As we show later (see Theorem 1), star-padding guarantees that we obtain the minimum distance.

3.2.2 Subsequence time warping matrix (STWM)

Star-padding is a good first step, and it can tell us (a) what the ending of the matching subsequence is, and (b) what its distance from the query sequence is. However, such applications often also need the starting time-tick of the match.

This is the motivation behind our second idea, the "*subsequence time warping matrix*" (STWM): we augment the time warping matrix and have each of its cells to record the starting position of each candidate subsequence. More specifically, the (t, i) cell of the usual time warping matrix contains the value $d(t, i)$, which is the best distance to match the prefix of length t from X with the prefix of length i from Y (i.e., $t = 1, \ldots, n$; $i = 1, \ldots, m$); our proposed STWM will also record $s(t, i)$, that is, the starting position corresponding to $d(t, i)$. In other words, the values $s(t, i)$ and $d(t, i)$ in the STWM mean that the subsequence from $s(t, i)$ through t gives distance $d(t, i)$, which is the best we can achieve for the t- and i- prefix of X and Y, respectively. We will give an arithmetical example of an STWM later (Figure 5).

Observation 2 *The subsequence time warping matrix (STWM) includes the distance value and starting position of each subsequence. Thus, we can identify the qualifying subsequence in a stream fashion.*

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.

In brief, from the above discussion we can summarize that our solution (i.e., the combination of star-padding

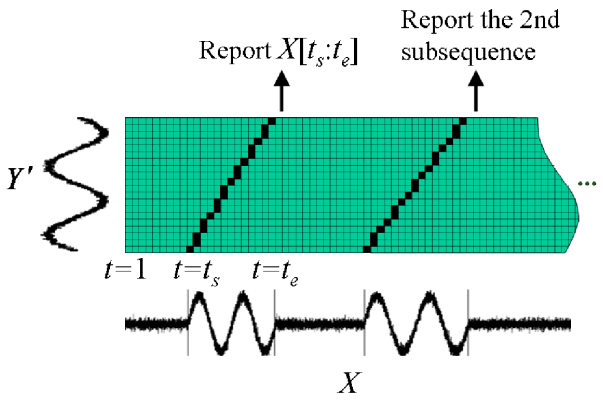

Figure 3. Illustration of SPRING. SPRING uses only a single matrix to capture all qualifying subsequences for disjoint queries.

and STWM) efficiently discards information about non-qualifying subsequences using a single matrix.

3.3. SPRING

In this section, we propose algorithms for solving the problems described in Section 3.1.2.

Our method, SPRING, efficiently detects high-similarity subsequences in data streams. Figure 3 illustrates how this is done. SPRING uses the STWM of X and Y', in which each element (t, i) retains both distance and starting position. SPRING reports all qualifying subsequences for disjoint queries, while giving the most similar subsequence $X[t_s : t_e]$ for best-match queries. Before introducing our algorithms, we give the details of the star-padding and STWM.

Given a sequence $Y = (y_1, \ldots, y_m)$, we have the star-padding of Y, i.e., $Y' = (y_0, y_1, \ldots, y_m)$ where $y_0 = (-\infty : +\infty)$. Let X be a sequence of length n, we can then derive the minimum distance $D(X[t_s : t_e], Y)$ from the matrix of X and Y'.

$$\begin{aligned} &D(X[t_s : t_e], Y) = d(t_e, m) = \min(d(t, m)) \\ &d(t, i) = \|x_t - y_i\| + d_{best} \\ &d_{best} = \min \begin{cases} d(t, i-1) \\ d(t-1, i) \\ d(t-1, i-1) \end{cases} \\ &d(t, 0) = 0, \;\; d(0, i) = \infty \\ &(t = 1, \ldots, n; i = 1, \ldots, m). \end{aligned} \tag{4}$$

As well as the distance $d(t, i)$, the matrix contains the starting position:

$$s(t, i) = \begin{cases} s(t, i-1) & (d(t, i-1) = d_{best}) \\ s(t-1, i) & (d(t-1, i) = d_{best}) \\ s(t-1, i-1) & (d(t-1, i-1) = d_{best}). \end{cases} \tag{5}$$

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

$$t_s = s(t_e, m). \tag{6}$$

The optimal warping path is obtained using the distance computation, and the starting position of the best subsequence is propagated through the matrix on the optimal warping path.

3.3.1 Algorithm

Let d_i and d'_i be arrays of m distance values, and let s_i and s'_i be arrays of m integers. The DTW distance of each subsequence can be incrementally computed as:

$$\begin{aligned} d_i &= \|x_t - y_i\| + d_{best} \\ d_{best} &= \min(d_{i-1},\ d'_i,\ d'_{i-1}) \\ &\quad d_0 = d'_0 = 0 \end{aligned} \tag{7}$$

where $d_i = d(t, i)$, and $d'_i = d(t-1, i)$ at time-tick t. Similarly, we obtain the starting position of the subsequence as:

$$s_i = \begin{cases} s_{i-1} & (d_{i-1} = d_{best}) \\ s'_i & (d'_i = d_{best}) \\ s'_{i-1} & (d'_{i-1} = d_{best}) \end{cases} \tag{8}$$

where $s_i = s(t, i)$, and $s'_i = s(t-1, i)$. Thus, we update m distance values and m integers for each time-tick.

The stream processing for best-match queries is straightforward; it simply uses Equations (7) and (8), and reports the best subsequence when the user requires it.

For disjoint queries, the most straightforward algorithm would be: as soon as we find a matching subsequence (i.e., with distance $\leq \epsilon$), we report it and then initialize the array of d_i. This algorithm satisfies the first condition of Problem 2 (i.e., $d_m \leq \epsilon$) and is useful if the user wants a quick response. This algorithm, however, does not satisfy the second condition of the problem. In fact, it may miss the optimal subsequence, if there are multiple overlapping subsequences within ϵ.

We introduce an new algorithm, which is carefully designed to (a) guarantee no false dismissals for the second condition of Problem 2 and (b) report each match as early as possible. As Figure 4 illustrates, for each incoming data point, we first incrementally update the distance d_i and determine the starting position s_i according to the computation of d_i. The algorithm reports the subsequence after confirming that the current optimal subsequence cannot be replaced by the upcoming subsequences. The idea is to keep track of the minimum distance, d_{min}, while investigating the group of overlapping subsequences. We report the subsequence that gives d_{min} when the arrays of d_i and s_i satisfy

$$\forall_i, d_i \geq d_{min} \vee s_i > t_e \tag{9}$$

which means that the captured optimal subsequence cannot be replaced by the upcoming subsequences. Otherwise, the

Algorithm SPRING
  input: a new value x_t at t
  output: qualifying subsequence if any
  for $i = 1$ to m **do**
    Compute d_i and s_i by Equations (7) (8);
  if $d_{min} \leq \epsilon$ **then**
    if $\forall_i, d_i \geq d_{min} \vee s_i > t_e$ **then**
      Report (d_{min}, t_s, t_e);
      // Reset d_{min} and the array of d_i
      $d_{min} = \infty$;
      for $i = 1$ to m **do**
        if $s_i \leq t_e$ **then**
          $d_i = \infty$;
    endif
  endif
  if $d_m \leq \epsilon \wedge d_m < d_{min}$ **then**
    $d_{min} = d_m$; $t_s = s_m$; $t_e = t$;
  endif
  Substitute d'_i for d_i;
  Substitute s'_i for s_i;

Figure 4. Algorithm for disjoint queries – prints the optimal subsequences.

upcoming candidate subsequences do not overlap with the captured optimal subsequence. We initialize d_{min} and the array of d_i after the output.

Here, we use Figure 5 to illustrate how the algorithm works.

Example 1 Assume that $\epsilon = 15$, $X = (5, 12, 6, 10, 6, 5, 13)$, and $Y = (11, 6, 9, 4)$. The element (t, i) of the matrix contains $d(t, i)$ and $s(t, i)$. At $t = 3$, we found candidate subsequence $X[2 : 3]$ whose distance $d(3, 4) = 14$ below ϵ. At $t = 4$, 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 $t = 5$. $X[2 : 5]$ is reported at $t = 7$ since we now know that none of the upcoming subsequences will be/is the optimal subsequence. Finally, because subsequences starting from $t = 7$ may be candidates for the next group, we do not initialize $d(7, 1)$.

4. Theoretical Analysis

Our upcoming experiments show that SPRING can efficiently spot qualifying subsequences. In this section, we do a theoretical analysis to demonstrate the accuracy and complexity of SPRING.

4.1. Accuracy

Theorem 1 *Given sequences X and Y, the DTW distance between X and Y' (i.e., the star-padding of Y) is the minimum distance between Y and all subsequences of X.*

$y_4 = 4$	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_t	5	12	6	10	6	5	13
t	1	2	3	4	5	6	7

Figure 5. Illustration of the proposed algorithm. The upper number shows the distance in each element of the matrix. The number in parentheses shows the starting position.

Proof: Let $X[t_s : t_e]$ be the subsequence that gives the minimum distance, then

$$f_{t_s}(t_e - t_s + 1, m) = \min(f_t(k, m))$$
$$(t = 1, \ldots, n; k = 1, \ldots, n - t + 1).$$

From $d(t, 0) = 0$, we have

$$d(t_s, 1) = f_{t_s}(1, 1).$$

Since the optimal warping path from element $(t_s, 1)$ through (t_e, m) gives the minimum distance, Equation (4) chooses the same warping path from $(t_s, 1)$. Thus, we have

$$d(t_e, m) = f_{t_s}(t_e - t_s + 1, m).$$

□

Lemma 1 *SPRING guarantees no false dismissals for best-match queries.*

Proof: By Theorem 1, the DTW distance between X and Y' is equal to the minimum distance between Y and subsequences of X. Since the subsequence time warping matrix contains the starting position, SPRING spots the subsequence that gives the minimum distance. □

Lemma 2 *SPRING guarantees no false dismissals for disjoint queries.*

Proof: Let d_{min} be the minimum distance computed from $X[t_s : t_e]$. At time-tick t ($t > t_e$), the overlapping subsequences give a larger distance if

$$\forall_i, d_i \geq d_{min}.$$

The upcoming candidate subsequences do not overlap with $X[t_s : t_e]$ if

$$\forall_i, s_i > t_e.$$

SPRING (See Figure 4) reports $X[t_s : t_e]$ only if

$$\forall_i, d_i \geq d_{min} \vee s_i > t_e.$$

Thus, it does not miss the optimal subsequence.

SPRING initializes d_i whose elements satisfy $s_i \leq t_e$ after reporting the optimal subsequence. Let $s(t,i) \leq t_e$. If the warping paths starting from $s(t,i)$ and passing through (t,i) give a distance that exceeds ϵ, the other subsequences passing through (t,i) also give a larger distance. Otherwise, all subsequences passing through (t,i) are included in the group of overlapping subsequences, which means that there is no need to report them. Thus, SPRING is guaranteed not to discard the upcoming candidate subsequences. □

4.2. Complexity

Let X be an evolving sequence of length n and Y be a sequence of fixed-length m.

Lemma 3 *The naive solution requires $O(nm)$ space and $O(nm)$ time per time-tick.*

Proof: The naive solution has to maintain $O(n)$ time warping matrices, and updates $O(nm)$ numbers every time-tick to identify qualifying subsequences. Thus, it requires $O(nm)$ time. Since the naive solution keeps two arrays of m numbers for each matrix, overall, it needs $O(nm)$ space. □

Lemma 4 *SPRING requires $O(m)$ space and $O(m)$ time per time-tick.*

Proof: SPRING keeps a single matrix, and updates $O(m)$ numbers every time-tick. Thus, SPRING requires $O(m)$ space and time. □

5. Experiments

To evaluate the effectiveness of SPRING, we carried out experiments on real and synthetic data sets. Our experiments were conducted on an Intel Xeon 2.8GHz with 1GB of memory, running Linux.

The experiments were designed to answer the following questions:

1. How successful is SPRING in capturing sequence patterns?
2. How does it scale with the sequence lengths n in terms of the computational time and memory space?
3. How well does SPRING handle multiple streams?

5.1. Discovery of sequence patterns

We present case studies on real and realistic data sets to demonstrate the effectiveness of our approach in discovering the qualifying subsequences for disjoint queries. Figure 6 shows how SPRING detects the qualifying subsequences. If multiple qualifying subsequences exist, we point them all out. Table 2 shows the details of the experimental results. In this table, 'Distance' means the DTW distance between the query sequence and each subsequence. 'Output time' indicates the time-tick at which SPRING reports the subsequence.

MaskedChirp

We used a synthetic data set, *MaskedChirp*, which consists of discontinuous sine waves with white noise. We varied the period of each disjoint sine wave in the sequence. We chose this setting because it resembles real data, such as voice data, which include sound and silent parts with varying time periods.

Figure 6 (a) shows that SPRING can perfectly identify all sound parts (i.e., the subsequences from #1 to #4) and that it is robust against noise.

Table 2 shows that the output time of each captured subsequence is very close to its end position. For example, the output time of subsequence #4 is 18844, which is close to its end position 18052 (=15171 + 2882 - 1), while our method guarantees the provision of the optimal subsequence. Note that the output time does not depend on threshold ϵ.

Temperature

We used temperature measurements (degrees Celsius) in the Critter data set, which comes from small sensors. The sensors give a reading approximately every minute. In this data set there are many missing values, which arise all the time. This is the same data set that was used previously [16].

As shown in Figure 6 (b), there are two similar patterns that significantly fluctuate with weather conditions (which range from 20 to 32 degrees). SPRING is not sensitive at all to the missing values. Actually, SPRING finds the days when the temperature fluctuates from cool to hot.

Kursk

The data set for Figure 6 (c) consists of seismic recordings from multiple sensors at different locations, which show the explosion of the Russian submarine Kursk [10] in 2000. Each sequence has single or multiple bursts.

The explosions shown in these sequences look similar; however, the intervals between large spikes are slightly different. This phenomenon was due to differences in environmental conditions such as underwater temperature. As can be seen in the figure, SPRING is not affected by the difference in the environmental conditions.

Sunspots

We know that sunspots appear in cycles. For example, during one 30-year period within the so-called "Maunder Minimum", only about 50 sunspots were observed, as opposed to the normal 40,000-50,000 spots. The average number of visible sunspots varies over time, increasing and decreasing

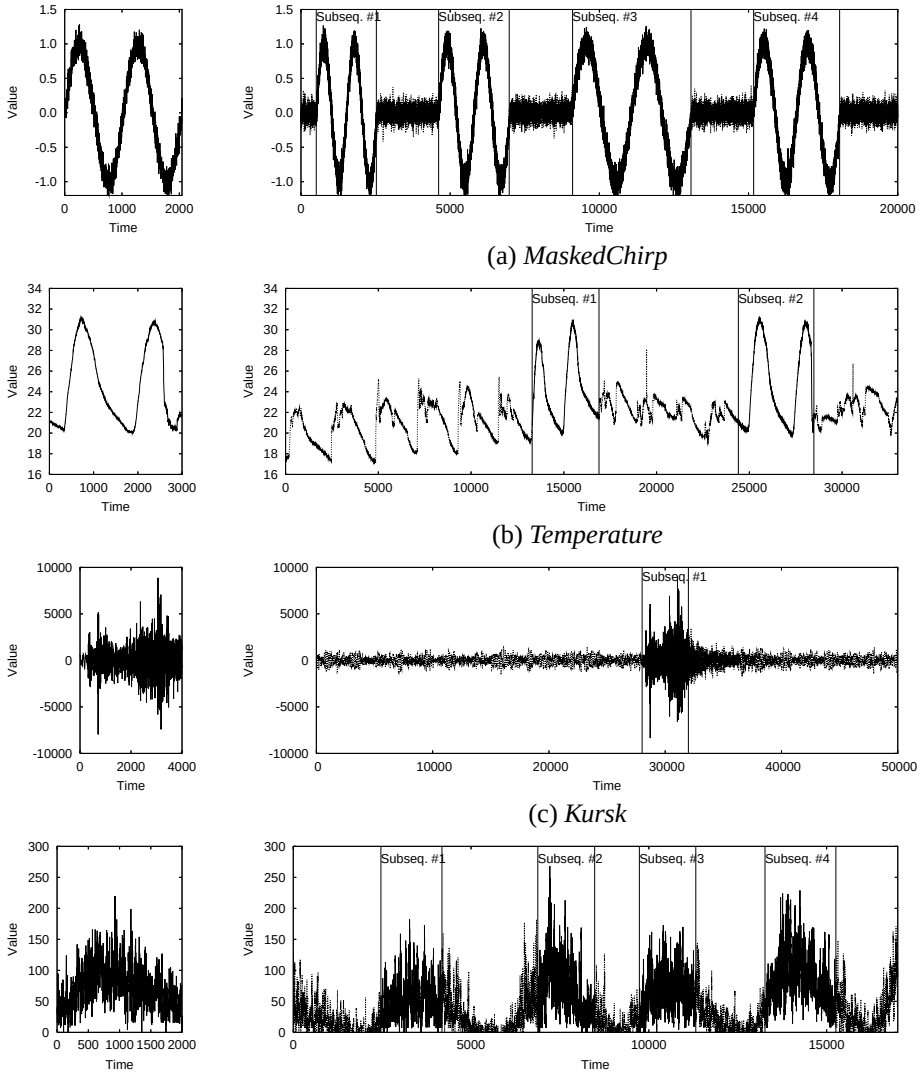

Figure 6. Discovery of sequence patterns in *MaskedChirp*, *Temperature*, *Kursk*, and *Sunspots*. The left and right columns show the query sequences and data sequences, respectively.

in a regular cycle of between 9.5 and 11 years, averaging about 10.8 years [1].

Each value in Figure 6 (d) indicates the number of sunspots per day. SPRING can capture bursty sunspot periods and identify the time-varying periodicity.

5.2. Performance

We did experiments to evaluate the efficiency and to verify the complexity of SPRING, which discussed in Section 4.2.

Figure 7 compares SPRING and the naive implementation in terms of computation time for varying sequence lengths n. Figure 8 shows the amount of memory space required to keep the time warping matrix (matrices). The plots were generated using *MaskedChirp*, which allowed us to control the sequence length. The length of the query sequence for these experiments was 256. The wall clock time is the average processing time needed to update the time warping matrix (matrices) for each time-tick and to capture the qualifying subsequences.

As we expected, SPRING identifies the qualifying subsequences much faster than the naive implementation (See Figure 7). The trend shown in the figure agrees with our theoretical discussion in Section 4.2. Compared with $O(nm)$, which the naive implementation requires, SPRING achieves a dramatic reduction in computation time: it requires constant time; i.e., it does not depend on n. In fact, SPRING is up to 650,000 times faster than the naive implementation.

SPRING is able to provide information about the ar-

[1] http://csep10.phys.utk.edu/astr162/lect/sun/sscycle.html

Table 2. Results of disjoint queries.

Data sets	Query sequences		Matching subsequences			
	Length	Threshold ϵ	Starting position	Length	Distance	Output time
MaskedChirp	2048	100	513	2015	10.05	3176
			4614	2366	11.39	7601
			9103	3969	18.59	14137
			15171	2882	12.42	18844
Temperature	3000	1000	13293	3602	820.1	17830
			24406	4073	6.5	28653
Kursk	4000	5.0e+9	28013	3981	7.06e+8	36711
Sunspots	2000	8.0e+5	2466	1717	5.67e+5	5591
			6878	1599	5.45e+5	9509
			9734	1587	4.87e+5	12257
			13266	1994	5.48e+5	16532

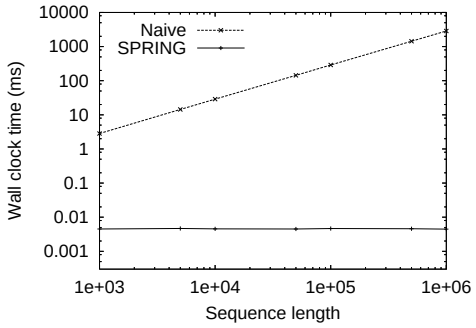

Figure 7. Wall clock time for disjoint queries as a function of sequence length. SPRING is up to 650,000 times faster.

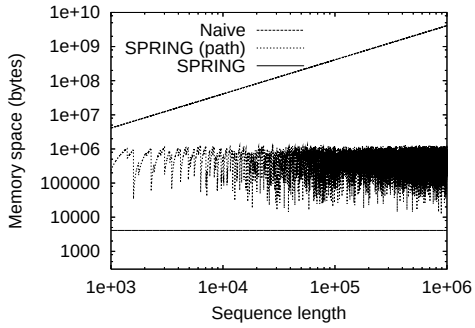

Figure 8. Memory space consumption for disjoint queries as a function of sequence length. SPRING can handle data streams with a small constant memory space.

rangement (i.e., the warping path) of the optimal subsequence and the query sequence although we assume that our method is required to keep track of the position of the optimal subsequence. In Figure 8, `SPRING(path)` indicates the space requirements of the former case, and `SPRING` shows that of the latter. The space requirement of `SPRING(path)` depends on the captured data. The figure, however, shows that it is clearly lower than that of the naive implementation. As we expected, `SPRING` needs a small constant space to keep track of the subsequence position, which shows a dramatic improvement.

5.3. Extension to multiple streams

We extend SPRING to handle multiple streams ("vector" streams), where each time-tick has not just a number, but a whole vector of k numbers, and the query is also a set of k sequences of m time-ticks. The driving application is motion capture data. A Motion Capture (or "mocap") sequence is created by recording motion information from a human actor while the actor is performing an action (e.g., walking, running, kicking). Special markers are placed on the joints of the actor (e.g., knees, hips, elbows), and their x-, y- and z-velocities are recorded, about 60 times per second. Eventually, a whole motion X is a time-evolving vector with k=62 dimensions and 60 samples per second, spanning several seconds.

The query Y is again a k-dimensional time sequence whose the goal is to find a matching subsequence within X. The intuition is the following: if Y is a walking motion, we want to find intervals in X that contain a walking-like motion.

We used a single sequence of 7 consecutive motions (See Figure 9), and other 4 sequences as query sequences, where each query sequence contains one of the 4 motions; walking, jumping, punching, and kicking. The data were from the CMU motion capture database [2]. We modified the algorithm of SPRING for the motion capture to report the starting and ending positions of the range of overlapping subsequences. SPRING perfectly captures all 7 motions shown in Figure 9 while still maintaining scalability. Due to the space limitations we omit the detailed experimental results.

[2] http://mocap.cs.cmu.edu/

(1) walking (2) jumping (3) walking (4) punching (5) walking (6) kicking (7) punching

Figure 9. Sequence of 7 motions.

6. Conclusions

We introduced the problem of subsequence matching under DTW, over data streams, and we propose SPRING, a new, fast algorithm to solve the problem. Notice that the DTW distance has been studied for finite, stored sequence sets (e.g., [8, 17, 21]). While we focus on stream processing in this paper, SPRING can obviously be applied to stored sequence sets, too, complementing the above solutions, and potentially making them faster.

In conclusion, SPRING has the following characteristics:

- It is fast and nimble: in contrast to the naive solution, SPRING requires only a single matrix to find the qualifying subsequences, and only constant space and time per time-tick; that is, it does not depend on the past length of data stream X.
- It guarantees no false dismissals.
- On real and realistic data, SPRING works as expected, discovering the qualifying subsequences quickly and accurately. Specifically, SPRING was up to 650,000 times faster.

References

[1] V. Ganti, J. Gehrke, and R. Ramakrishnan. Mining data streams under block evolution. *SIGKDD Explorations*, 3(2):1–10, 2002.

[2] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *Proceedings of VLDB*, pages 79–88, Rome, Italy, Sept. 2001.

[3] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams: Theory and practice. *IEEE TKDE*, 15(3):515–528, 2003.

[4] P. Indyk, N. Koudas, and S. Muthukrishnan. Identifying representative trends in massive time series data sets using sketches. In *Proceedings of VLDB*, pages 363–372, Cairo, Egypt, September 2000.

[5] J.-S. R. Jang and H.-R. Lee. Hierarchical filtering method for content-based music retrieval via acoustic input. In *Proceedings of ACM Multimedia*, pages 401–410, September/October 2001.

[6] W. Johnson and J. Lindenstrauss. Extensions of lipschitz mappings into hilbert space. *Contemporary Mathematics*, pages 26:189–206, 1984.

[7] H. Kawasaki, T. Yatabe, K. Ikeuchi, and M. Sakauchi. Automatic modeling of a 3d city map from real-world video. In *Proceedings of ACM Multimedia (1)*, pages 11–18, October/November 1999.

[8] E. J. Keogh. Exact indexing of dynamic time warping. In *Proceedings of VLDB*, pages 406–417, Hong Kong, China, August 2002.

[9] S.-W. Kim, S. Park, and W. W. Chu. An index-based approach for similarity search supporting time warping in large sequence databases. In *Proceedings of ICDE*, pages 607–614, April 2001.

[10] K. Koper, T. Wallace, S. Taylor, and H. Hartse. Forensic seismology and the sinking of the kursk. *EOS Trans., AGU, 82*, pages 37,45–46, 2001.

[11] D. W. Mount. *Bioinfomatics: Sequence and Genome Analysis*. Cold Spring Harbor, New York, 2000.

[12] K. Otsuka, T. Horikoshi, S. Suzuki, and H. Kojima. Memory-based forecasting for weather image patterns. In *Proceedings of the 17th Conference on Artificial Intelligence (AAAI)*, pages 330–336, July 2000.

[13] S. Papadimitriou, A. Brockwell, and C. Faloutsos. Adaptive, hands-off stream mining. In *Proceedings of VLDB*, pages 560–571, Berlin, Germany, Sept. 2003.

[14] S. Papadimitriou, J. Sun, and C. Faloutsos. Streaming pattern discovery in multiple time-series. In *Proceedings of VLDB*, pages 697–708, Trondheim, Norway, August-September 2005.

[15] L. Rabiner and B.-H. Juang. *Fundamentals of Speech Recognition*. Englewood Cliffs, N. J., 1993.

[16] Y. Sakurai, S. Papadimitriou, and C. Faloutsos. Braid: Stream mining through group lag correlations. In *Proceedings of ACM SIGMOD*, pages 599–610, Baltimore, Maryland, June 2005.

[17] Y. Sakurai, M. Yoshikawa, and C. Faloutsos. Ftw: Fast similarity search under the time warping distance. In *Proceedings of PODS*, pages 326–337, Baltimore, Maryland, June 2005.

[18] T. S. F. Wong and M. H. Wong. Efficient subsequence matching for sequence databases under time warping. In *IDEAS*, pages 139–148, Sept. 2003.

[19] B.-K. Yi, H. V. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. In *Proceedings of ICDE*, pages 201–208, February 1998.

[20] Y. Zhu and D. Shasha. Statistical monitoring of thousands of data streams in real time. In *Proceedings of VLDB*, pages 358–369, Hong Kong, China, Aug. 2002.

[21] Y. Zhu and D. Shasha. Warping indexes with envelope transforms for query by humming. In *Proceedings of ACM SIGMOD*, pages 181–192, San Diego, California, June 2003.