Fast and Exact Monitoring of Co-evolving Data Streams

Yasuko Matsubara
Kumamoto University
yasuko@cs.kumamoto-u.ac.jp

Yasushi Sakurai
Kumamoto University
yasushi@cs.kumamoto-u.ac.jp

Naonori Ueda
NTT Communication Science Labs
ueda.naonori@lab.ntt.co.jp

Masatoshi Yoshikawa
Kyoto University
yoshikawa@i.kyoto-u.ac.jp

***Abstract*—Given a huge stream of multiple co-evolving sequences, such as motion capture and web-click logs, how can we find meaningful patterns and spot anomalies? Our aim is to monitor data streams statistically, and find subsequences that have the characteristics of a given hidden Markov model (HMM). For example, consider an online web-click stream, where massive amounts of access logs of millions of users are continuously generated every second. So how can we find meaningful building blocks and typical access patterns such as weekday/weekend patterns, and also, detect anomalies and intrusions?**

In this paper, we propose *StreamScan*, a fast and exact algorithm for monitoring multiple co-evolving data streams. Our method has the following advantages: (a) it is effective, leading to novel discoveries and surprising outliers, (b) it is exact, and we theoretically prove that *StreamScan* guarantees the exactness of the output, (c) it is fast, and requires $O(1)$ time and space per time-tick. Our experiments on 67GB of real data illustrate that *StreamScan* does indeed detect the qualifying subsequence patterns correctly and that it can offer great improvements in speed (up to 479,000 times) over its competitors.

I. Introduction

Data streams naturally arise in countless domains, such as medical analysis [10], online text [7], social activity mining [17], and sensor network monitoring [12]. For example, consider an online web-click stream, where a huge collection of logging entries are generated every second, with information of millions of users and URLs. The web-site owners would like to detect intrusions or target designed advertisements by investigating the user-click patterns. In such a situation, the most fundamental requirement is the efficient monitoring of data streams. Since the data streams arrive online at high bit rates and are potentially unbounded in size, the algorithm should handle 'big data streams' of billions (or even trillions [28]) of entries with fast response times, that is, it cannot afford any post-processing. And in addition, since the sampling rates of streams are frequently different and their time periods vary in practical situations, the mechanism should be robust against noise and provide scaling of the time axis.

The hidden Markov model (HMM) is a ubiquitous tool for representing probability distributions over sequences of observations. Inspired by statistical approaches, a vast amount of work has been undertaken on HMMs. HMMs have become the method of choice for modeling stochastic processes and sequences in applications including speech recognition [31] and sensor analysis [12].

A. Motivation and challenges

In this paper, we address the problem of efficiently monitoring data streams with HMMs. Informally, the problem we want to solve is defined as follows:

Given *a data stream X and a query model $\mathbf{\Theta}$,* ***Find*** *the subsequences that have the characteristics of $\mathbf{\Theta}$.*

Note that we can also consider a case where we have multiple queries (i.e., $\mathbf{\Theta}_1, \mathbf{\Theta}_2, \dots$). We continue the discussion focusing on a single query for simplicity throughout this paper, our algorithm however can be applied to multiple queries.

We present the main intuition and motivation with a real example. Figure 1 shows the original data stream taken from motion capture data. It is composed of several consecutive motions, such as "walking", "squatting" and "running" (Figure 1(a)), and each motion consists of four co-evolving sequences: left/right arms, and left/right legs (Figure 1(b)). Assume that we have a stream of sequences of these motions, and we wish to find the specific motions (e.g., "walking" and "running") from the stream. Our algorithm is able to discover subsequences, each of which has the characteristics of "walking", "running" and "punching" according to a given model. Most importantly, our algorithm guarantees the exactness of the results for data stream monitoring, which means that it perfectly captures the qualifying subsequences and the locations of their cut points without any loss of accuracy.

B. Contrast with competitors

The HMM is one of the most useful techniques for the statistical characterization of time-varying dynamic patterns. Many algorithms have been proposed for monitoring data streams in an online fashion. However, relatively little attention has been paid to monitoring data streams through HMMs. For example, *SMM* (Wilpon et al. [36]) is capable of finding the best match of the HMM with a segment of the sequence. It is based on the sliding window approach, and it takes $O(n^2)$ time to scan the entire sequence with $O(n)$ space, where n is the sequence length. Silaghi proposed *SFR* [31], which is a more efficient algorithm for keyword spotting and segmentation. It still requires $O(m)$ time and space to find appropriate subsequences, where m is the length of qualifying subsequences/segments. Also note that the above methods are not designed to monitor big data streams in an online fashion.

By contrast, our algorithm not only works continuously in a streaming fashion, but also provides vastly better performance

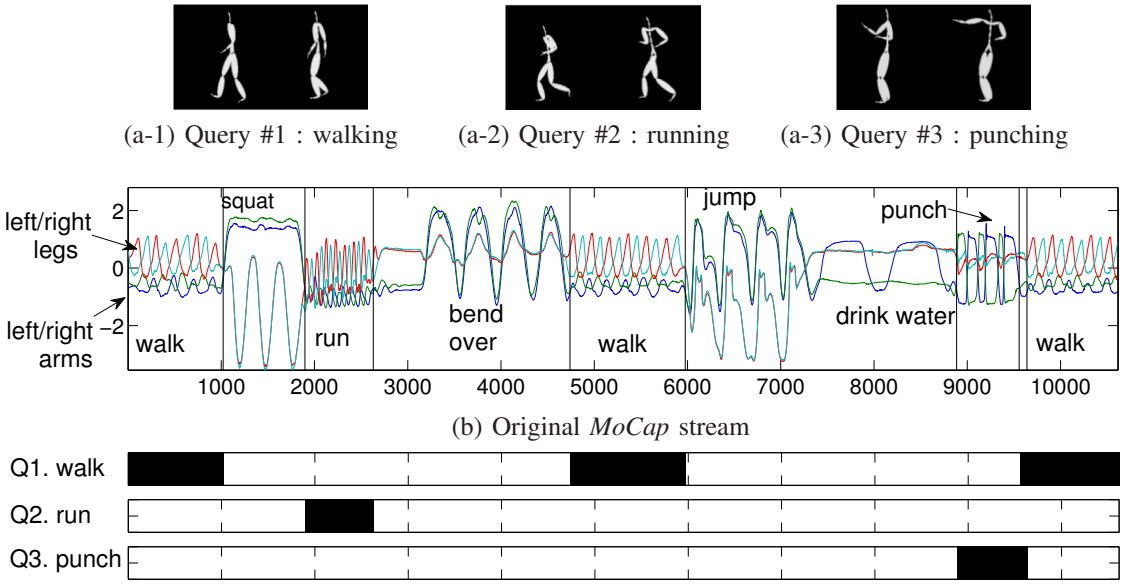


Fig. 1. Illustration of stream monitoring with *StreamScan*. Given query models (e.g., walking, running and punching motions), *StreamScan* incrementally identifies the specific motions, and also finds the locations of all their cut points from the stream.

than the alternative solutions in terms of speed and memory. The performance of our system does not depend on the past length of the data stream, or the length of each segment.

C. Contributions

We present a new online method, *StreamScan*, for sequence pattern discovery in data streams. Our method has the following advantages:

1) **Effectiveness:** *StreamScan* operates on a huge collection of time-series, and detects fruitful subsequences from data streams.
2) **Exactness:** we discuss the accuracy of *StreamScan* theoretically, and show that it guarantees no false dismissals (i.e., it does not sacrifice accuracy).
3) **Scalability:** the computation time and memory consumption do not depend on the length n of data streams. There is no need to revisit expiring data streams. We perform extensive experiments with real data and show that *StreamScan* can be up to 479,000 times faster than competitors.

D. Outline

The rest of the paper is organized as follows. Section II describes related work, followed by definitions in Section III. We then describe our method in Section IV. Section V discusses the theoretical analysis. Section VI show our experimental results on a variety of datasets. We conclude in Section VII.

II. Related Work

Related work falls broadly into two categories: (1) hidden Markov models and (2) pattern discovery in time-series. We provide a survey of the related literature.

A. Hidden Markov models

Hidden Markov models have recently been used in various research areas including speech recognition [36], [37], [31]. Wilpon et al. [36], [37] focus on the problem of keyword spotting in unconstrained speech using HMMs, and presented a novel method for finding the best match of the keyword model with a segment of the observation, while Silaghi [31] extended the work in [36] and proposed a more efficient algorithm, namely, Segmentation by Filler Re-estimation (SFR). Clearly their focus is on stored datasets, as opposed to data streams. Our streaming algorithm guarantees to detect the best subsequence without any loss while it achieves a large reduction in terms of time and space. As regards HMM-based approaches for large time-evolving sequences, [12] presented a system for executing event queries over Markovian streams, generated from RFID sensors, while [5] proposed an exact and efficient search algorithm for large HMM datasets.Very recently, Wang et al. [35] presented a pattern-based hidden Markov model (pHMM), which is a new dynamical model for time-series segmentation and clustering, while, [19] developed a fully-automatic mining algorithm for co-evolving time-series. Most importantly, they are based on the iterative process, that is, they are not capable of online processing.

B. Pattern discovery in time series

In recent years, there has been an explosion of interest in mining time series [2], [3], [25], [28], [33], [18]. Traditional approaches applied to data mining include auto regression (AR) and variations [13], or linear dynamical systems (LDS), Kalman filters (KF) and variants [8], [14], [15]. Similarity search, indexing and pattern discovery in time sequences have also attracted huge interest [4], [9], [6], [27], [11], [25], [16], [34], [26], [22]. Although none of the streaming methods deals with HMMs, we review them here because they examine

TABLE I. SYMBOLS AND DEFINITIONS.

Symbol	Definition
n	Number of time-ticks
X	Data stream of length n
$\boldsymbol{x}_t$	d-dimensional vector of X at time-tick $t = 1, \ldots, n$
$X[t_s : t_e]$	Subsequence of X, including values in positions t_s through t_e
m	Length of $X[t_s : t_e]$, i.e., $m = t_e - t_s + 1$
k	Number of hidden states
$\boldsymbol{\Theta}$	Set of parameters governing the model
$\boldsymbol{\pi} = \{\pi_i\}$	Initial state probability in state i
$\mathbf{A} = \{a_{ij}\}$	State transition probability from state i to j
$\mathbf{B} = \{b_i(v)\}$	Probability of symbol v in state i
ϵ	Threshold for finding qualifying subsequences
δ	Threshold of subsequence length
$P(X, \boldsymbol{\Theta})$	Likelihood function of X given $\boldsymbol{\Theta}$
$p_i(t)$	Probability of element (t, i), i.e., state i at time-tick t
$v_i(t)$	Cumulative likelihood of (t, i)
$s_i(t)$	Starting position of (t, i)

related topics, such as pattern discovery, summarization, and lossy compression for data streams. The work in [32] presented an approach for recursively predicting motion sequence patterns. For web-click and social analysis, Agarwal et al. [1] exploit the Gamma-Poisson model to estimate "click-through rates" in the context of content recommendation, while the work in [21] studied the rise and fall patterns in information diffusion process through online social medias. TriMine [20] is a scalable method for forecasting complex time-series. Regarding to the stream monitoring, SPIRIT [24] addresses the problem of capturing correlations and finding hidden variables corresponding to trends in collections of co-evolving data streams. Sakurai et al. [30] introduced an approximation technique, BRAID, proposed BRAID, which efficiently detects lag correlations between data streams. SPRING [29] is an efficient algorithm for monitoring multiple numerical streams under the dynamic time warping (DTW) distance. Very recently, Mueen et al. studied time series motifs [23], while Rakthanmanon et al. [28] proposed a novel algorithm designed to accelerate a similarity search for trillions of time-series under the DTW distance. However, none of the above methods examines pattern discovery on streams with the HMMs.

III. PROBLEM FORMULATION

In this section, we define the problems and some fundamental concepts.

A. Hidden Markov model

The hidden Markov model (HMM) is a statistical model where the system being modeled is assumed to be a Markov process with unknown (i.e., "hidden") states. An HMM parameter set, $\boldsymbol{\Theta} = \{\pi, \mathbf{A}, \mathbf{B}\}$, is composed of the following probabilities:

- *Initial state probability*: $\boldsymbol{\pi} = \{\pi_i\}_{i=1}^{k}$,
- *State transition probability*: $\mathbf{A} = \{a_{ij}\}_{i,j=1}^{k}$,

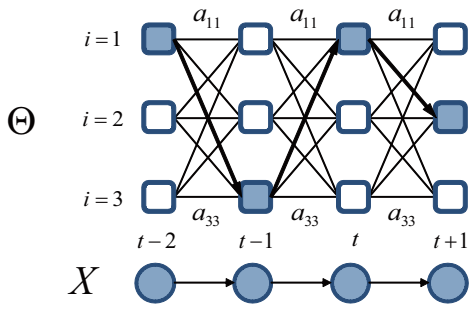


Fig. 2. Illustration of the HMM trellis structure. Given a sequence X and a model $\boldsymbol{\Theta}$ (here, hidden states $k = 3$, sequence length $n = 4$), the shaded states in the structure denote the Viterbi path.

- *Output probability*: $\mathbf{B} = \{b_i(\boldsymbol{x})\}_{i=1}^{k}$ [1].

Given a model $\boldsymbol{\Theta}$ and an input sequence X, the likelihood value $P(X, \boldsymbol{\Theta})$ is computed as follows:

$$P(X, \boldsymbol{\Theta}) = \max_{1 \le i \le k} \{p_i(n)\} \tag{1}$$

$$p_i(t) = \begin{cases} \pi_i b_i(\boldsymbol{x}_1) & (t = 1) \\ \max_{1 \le j \le k} \{p_j(t-1) a_{ji}\} b_i(\boldsymbol{x}_t) & (2 \le t \le n) \end{cases}$$

where $p_i(t)$ is the maximum probability of state i at time-tick t. The likelihood is computed based on the "*trellis structure*" shown in Figure 2, where states lie on the vertical axis, and sequences are aligned along the horizontal axis. The likelihood is computed using a dynamic programming approach, called a Viterbi algorithm, which maximizes the probabilities from previous states (i.e., each state probability is computed using all previous state probabilities, associating transition probabilities, and output probabilities). The state sequence, which gives the likelihood, is called the Viterbi path. The Viterbi algorithm generally requires $O(nk^2)$ time since it compares k transitions to obtain the maximum probability for every state, that is, the trellis structure consists of $(n \times k)$ elements. Note that the space complexity is $O(k)$ since the algorithm needs only two columns (i.e., the current and previous columns) of the trellis structure to compute the likelihood.

Example 1: Assume the following model and sequence.

$$\pi = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \mathbf{A} = \begin{bmatrix} 0.5 & 0.5 & 0 \\ 0.5 & 0.25 & 0.25 \\ 0 & 0 & 1 \end{bmatrix},$$
$$\mathbf{B} = \begin{bmatrix} 1 & 0 & 0 \\ 0.75 & 0.25 & 0 \\ 0 & 0 & 1 \end{bmatrix}, X = (1, 1, 2, 3).$$

From the Viterbi algorithm, we have

$$\begin{array}{llll} p_1(1){=}1, & p_1(2){=}0.5, & p_1(3){=}0, & p_1(4){=}0 \\ p_2(1){=}0, & p_2(2){=}0.75{\cdot}0.5, & p_2(3){=}0.5^2{\cdot}0.25, & p_2(4){=}0 \\ p_3(1){=}0, & p_3(2){=}0, & p_3(3){=}0, & p_3(4){=}0.5^2{\cdot}0.25^2. \end{array}$$

The state sequence (u_1, u_1, u_2, u_3) gives the maximum probability. Consequently, we have $P(X, \boldsymbol{\Theta}) = (0.5)^2 \cdot (0.25)^2$.

[1] In this paper, we mainly focus on numerical sequences, and we assume a Gaussian distribution for the output probability, (i.e., $\mathbf{B} = \{\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_i, \boldsymbol{\sigma}_i^2)\}_{i=1}^{k}$). However, our solution, *StreamScan* can handle any other types of categorical and numerical distributions. Our algorithm is completely independent of such choice.

B. Problem definition

A data stream X is a semi-infinite sequence of d-dimensional vectors, $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n, \ldots\}$, where $\boldsymbol{x}_n$ is the most recent vector. Let $X[t_s : t_e]$ denote the subsequence starting from time-tick t_s and ending at t_e ($1 \leq t_s \leq t_e \leq n$). Our goal is to find a subsequence $X[t_s : t_e]$ that has a high likelihood value with respect to a given model $\boldsymbol{\Theta}$. So, what are the real requirements for monitoring data streams? We need a one-path algorithm that reports all the qualifying subsequences, immediately, at any point in time, while discarding redundant information. In short, the ideal solution would satisfy all the following requirements.

- **Exponential threshold function:** The likelihood decreases as the subsequence length grows since it is a multiplication of the state probabilities in the trellis structure. Therefore, the likelihood threshold should be an exponential function of the subsequence length m, and so we set it at ϵ^m. More concretely, we want to identify subsequences that satisfy $P(X[t_s : t_e], \boldsymbol{\Theta}) \geq \epsilon^m$, where m is the length of $X[t_s : t_e]$ (i.e., $m = t_e - t_s + 1$).
- **Minimum length of subsequence matches:** In practice, we might detect very short and "meaningless" subsequences. However, this is insufficient for many real applications. We thus introduce a concept, namely, the minimum length of subsequence matches, to enable us to discard such meaningless subsequences and to detect the optimal subsequences that satisfy 'real' user requirements. Specifically, we want to satisfy $P(X[t_s : t_e], \boldsymbol{\Theta}) \geq \epsilon^{m-\delta}$. Here, the minimum length δ should be provided by users. We detect subsequences whose lengths exceed δ.
- **Non-overlapping matches:** Whenever the query $\boldsymbol{\Theta}$ matches a subsequence of X, we expect there to be several other matches by subsequences that heavily overlap the "local maximum" best match. 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". Thus, we propose adding one more condition designed to discard all these extra matches. Specifically, overlapping matches are defined as subsequences whose Viterbi paths cross, that is, share at least one element in the trellis structure. We shall use the term "optimal" subsequence hereafter, to denote exactly the subsequence that is the local best, among the set of overlapping subsequences of X.

Consequently, the main problem we propose and solve is as follows:

Problem 1: Given a stream X, a model $\boldsymbol{\Theta}$, and thresholds ϵ and δ, report all subsequences $X[t_s : t_e]$ such that

1) the subsequences are appropriate for $\boldsymbol{\Theta}$; that is, $P(X[t_s : t_e], \boldsymbol{\Theta}) \geq \epsilon^{m-\delta}$
2) among several overlapping matches, report only the local maximum, i.e.,

$$P(X[t_s : t_e], \boldsymbol{\Theta}) \cdot \epsilon^{\delta - m} \tag{2}$$

is the largest value in the group of overlapping subsequences that satisfy the first condition.

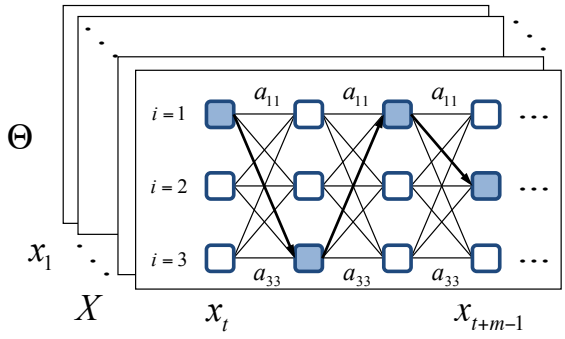


Fig. 3. Illustration of subsequence identification with the sliding model method (*SMM*). It maintains the trellis structures, starting from every time-tick $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t, \ldots)$.

Our 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. Also, we mainly focus on a single query for simplicity, however, our method can be easily applied to the case of multiple queries.

C. Previous solutions

For Problem 1, if the user requires a theoretical guarantee (i.e., no loss of accuracy), the most straightforward (and slowest) solution would be to consider all the possible subsequences $X[t_s : t_e]$ ($1 \leq t_s \leq t_e \leq n$) and compute the likelihood of each subsequence of X. This method requires $O(n^2)$ trellis structures, thus the time complexity would be $O(n^3k^2)$ (or $O(n^2k^2)$ per time-tick). This method is extremely expensive, and cannot be extended to the streaming case.

Wilpon et al. [36] proposed a much better solution, namely, the "sliding model method" (*SMM*). To find a qualifying subsequence $X[t_s : t_e]$, it computes the likelihood based on the trellis structure starting from every time-tick (see Figure 3) and applies the Viterbi algorithm. We can apply this solution to our problem. Let $p_{s,i}(t)$ be the probability of the i-th state at time t in the s-th trellis structure, which starts from time s. The maximum likelihood of the subsequence matching given X and $\boldsymbol{\Theta}$ can be obtained as follows:

$$P(X[t_s : t_e], \boldsymbol{\Theta}) = \max_{1 \leq i \leq k} \{p_{t_s,i}(t_e - t_s + 1)\} \tag{3}$$

$$p_{s,i}(t) = \begin{cases} \pi_i b_i(x_t) & (t = s) \\ \max_{1 \leq j \leq k}\{p_{s,j}(t-1)a_{ji}\}b_i(x_t) & (s < t \leq t_e), \end{cases}$$

$$(s = 1, \ldots, n; t = 1, \ldots, n - s + 1; i = 1, \ldots, k).$$

We then determine the appropriate subsequence in which Equation 2 is the maximum value in each overlapping group. This solution also guarantees no false dismissal. Since this solution needs $O(n)$ trellis structures, $O(nk^2)$ numbers have to be computed for each time-tick (also, the total computation cost is $O(n^2k^2)$).

More recently, Silaghi proposed a more efficient algorithm, *SFR* [31], which is based on *SMM*, but it has an implicit filler

state. For each structure, if the likelihood value in Equation 2 is not improved, (i.e., reaches the maximum value), it stops the computation. *SFR* requires $O(m)$ trellis structures, because it needs to retain all possible pairs of overlapping candidate subsequences.

IV. Proposed Method

In this section, we provide a new algorithm, namely, *StreamScan*, for dealing with Problem 1.

A. Fundamental concept

Our solution is based on the the following ideas.

Approach 1 (Cumulative likelihood function): We introduce the *cumulative likelihood function*: $V(X[t_s : t_e], \mathbf{\Theta})$ (see Equation 4), which requires only a single trellis structure to find the optimal subsequences of X.

Instead of creating a new trellis structure for every time-tick, which needs $O(n)$ structures, the cumulative likelihood function requires only a single structure, and thus it greatly reduces both time and space. As we show later (see Lemma 1), this function guarantees that we obtain the best subsequences of X.

Although important, this function is not immediately applicable to our problems. The cumulative likelihood function is a good first step, and it can tell us the end position of the matching subsequence. However, users and applications also often need the starting time-tick of the match and the likelihood of the matching subsequence. This is the motivation behind our second idea.

Approach 2 (Subsequence trellis structure): We propose keeping the starting position of each subsequence as well as the probability/likelihood. Our streaming algorithm detects the qualifying subsequences efficiently by using the subsequence trellis structure (*STS*). Thus, we can identify the qualifying subsequence in a stream fashion.

We augment the trellis structure to have each of its cells record the starting position of each candidate subsequence. More specifically, the i-th state at time t of the usual trellis structure contains the value $v_i(t)$, which is the highest cumulative likelihood to match the t-th value of X with the i-th state of $\mathbf{\Theta}$ (i.e., $t = 1, \ldots, n$; $i = 1, \ldots, k$); our proposed *STS* will also record $s_i(t)$, that is, the starting position corresponding to $v_i(t)$. In other words, the values $s_i(t)$ and $v_i(t)$ in the structure mean that the subsequence from $s_i(t)$ through t gives $v_i(t)$, which is the best we can achieve for the (t, i) element of the structure. The technical advantage is the output exactness. We carefully design our algorithm to detect all qualifying subsequences and discards information about non-qualifying subsequences safely and efficiently. Details are described in the next subsection.

B. StreamScan

We now propose a one-path algorithm for solving the problems described in Section III-B. Our method, *StreamScan*, efficiently detects appropriate subsequences in data streams. Figure 4 illustrates how this is done. *StreamScan* uses *STS*, in which each element (t, i) (i.e., the t-th value of X and the i-th state of $\mathbf{\Theta}$) retains both cumulative likelihood and starting position. *StreamScan* reports all qualifying subsequences.

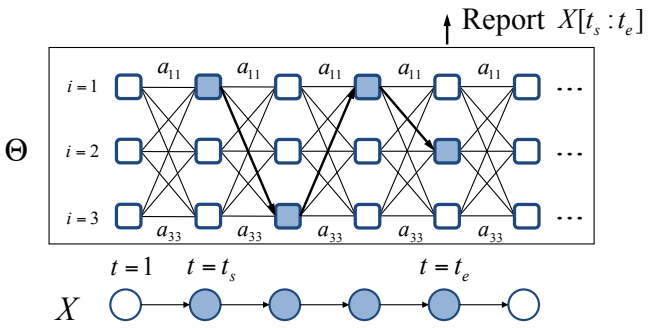


Fig. 4. Illustration of *StreamScan*. Given a stream X and a model $\mathbf{\Theta}$, *StreamScan* uses only a single trellis structure to capture all qualifying subsequences ($X[t_s : t_e]$).

Given a sequence X of length n, we can derive the cumulative likelihood value $V(X[t_s : t_e], \mathbf{\Theta})$ of $X[t_s : t_e]$ as follows:

$$V(X[t_s : t_e], \mathbf{\Theta}) = v_{best}(t_e) = \max_{1 \le i \le k} \{v_i(t_e)\} \tag{4}$$

$$v_i(t) = \max \begin{cases} \pi_i b_i(x_t) \cdot \epsilon^{-1} \\ \max_{1 \le j \le k} \{v_j(t-1) a_{ji}\} b_i(x_t) \cdot \epsilon^{-1} \end{cases}$$

$$(t = 1, \ldots, n; i = 1, \ldots, k).$$

As well as $v_i(t)$, the structure contains the starting position:

$$s_i(t) = \begin{cases} (t, i) & (v_i(t) = \pi_i b_i(x_t) \cdot \epsilon^{-1}) \\ s_j(t-1) & (v_i(t) \neq \pi_i b_i(x_t) \cdot \epsilon^{-1} \\ & \quad \wedge v_j(t-1) = v_{best}(t-1)). \end{cases} \tag{5}$$

The optimal path is obtained using the likelihood computation, and the starting position of the optimal subsequence is propagated through the structure on the path. The likelihood of the subsequence is obtained from the cumulative likelihood and the subsequence length as follows:

$$P(X[t_s : t_e], \mathbf{\Theta}) = V(X[t_s : t_e], \mathbf{\Theta}) \cdot \epsilon^m, \tag{6}$$

where m is the subsequence length (i.e., $m = t_e - t_s + 1$).

Algorithm. We introduce a new algorithm, which we carefully designed to (a) guarantee no false dismissals for the second condition of Problem 1 and (b) report each match as early as possible. As Algorithm 1 illustrates, for each incoming data point, we first incrementally update the cumulative likelihood $v_i(t)$ and determine the starting position $s_i(t)$ according to the computation of $v_i(t)$.

The candidate set $\mathcal{S}$ includes multiple optimal subsequences with different starting positions and retains the information on each candidate C (i.e., the cumulative likelihood C_v, the starting position C_s, and the end position C_e). The idea is to keep track of the maximum value, C_v, among the cumulative likelihood values of overlapping subsequences. We report the subsequence that gives C_v when C_s satisfies:

$$\forall_i, C_s \neq s_i(t), \tag{7}$$

Algorithm 1 StreamScan ($\boldsymbol{x}_t$)

Input: a new vector $\boldsymbol{x}_t$ at time-tick t
Output: a qualifying subsequence if any
(i.e., C_p: likelihood, C_s: starting position, C_e: end position)
for $i = 1$ to k **do**
  // Cumulative likelihood derived by Equation (4)
  Compute $v_i(t)$;
  // Starting position derived by Equation (5)
  Compute $s_i(t)$;
  $e_i(t) := (t, i)$; // End position
  if $v_i(t) \geq \epsilon^{-\delta}$ **then**
    if $s_i(t) \notin \mathcal{S}$ **then**
      // Add the subsequence into the candidate set $\mathcal{S}$
      Add $v_i(t)$, $s_i(t)$, and $e_i(t)$ to $\mathcal{S}$;
    else
      for each candidate $C \in \mathcal{S}$ **do**
        // Update the maximum cumulative likelihood and end position
        if $s_i(t) = C_s \ \wedge \ v_i(t) \geq C_v$ **then**
          $C_v := v_i(t)$; $C_e := e_i(t)$;
        end if
      end for
    end if
  end if
end for
// Report the optimal subsequence
for each candidate $C \in \mathcal{S}$ **do**
  if $\forall_i, C_s \neq s_i(t)$ **then**
    // Compute the likelihood of the subsequence, C_p
    $C_p = C_v \cdot \epsilon^l$;
    Report (C_p, C_s, C_e);
    Remove C from $\mathcal{S}$;
  end if
end for

which means that the captured optimal subsequence cannot be replaced by the upcoming subsequences. Otherwise, the upcoming candidate subsequences do not overlap the captured optimal subsequence. Finally, we compute the likelihood C_p of the output subsequence and then report it in stream processing.

Example 2: Here, we use Figure 5 to illustrate how the algorithm works. Assume that $\epsilon = 0.1$, $\delta = 3$, and the following model and sequence. [2]

$$\pi = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \mathbf{A} = \begin{bmatrix} 0.5 & 0.5 & 0 \\ 0.5 & 0.25 & 0.25 \\ 0 & 0 & 1 \end{bmatrix},$$
$$\mathbf{B} = \begin{bmatrix} 1 & 0 & 0 \\ 0.75 & 0.25 & 0 \\ 0 & 0 & 1 \end{bmatrix}, X = (3, 1, 1, 2, 3, 3, 3, 1).$$

The element (t, i) of the score matrix contains $v_i(t)$ and $s_i(t)$. The shaded elements denote the optimal Viterbi path. At $t = 1$, the algorithm begins to compute the cumulative likelihood value for each state. At $t = 6$, we found candidate subsequence $X[2:6]$ whose likelihood value is $v_3(6) = 1562.5$. Although the likelihood value is larger than $\epsilon^{-\delta}$, we do not report $X[2:6]$ since this can be replaced by the upcoming subsequence. We then capture the optimal subsequence $X[2:7]$ at $t = 7$. Finally, $X[2:7]$ is reported at $t = 8$ since we now know that none of the upcoming subsequences is/will be the optimal subsequence.

[2] For simplicity, we assume a discrete sequence for X, where the output probability $\mathbf{B}$ is a categorical distribution.

V. Theoretical Analysis

In this section, we undertake a theoretical analysis to demonstrate the accuracy and complexity of *StreamScan*.

A. Accuracy

Lemma 1: Given a sequence X and a model $\boldsymbol{\Theta}$, Problem 1 is equivalent to the following conditions:

1) $V(X[t_s : t_e], \boldsymbol{\Theta}) \geq \epsilon^{-\delta}$
2) $V(X[t_s : t_e], \boldsymbol{\Theta}) \cdot \epsilon^{\delta}$ is the maximum value in each group of overlapping subsequences.

Proof: Let us assume that the Viterbi path of $X[t_s : t_e]$ starts from the u-th state at time t_s (i.e., the element (t_s, u)) in the trellis structure. From Equation 3 and Equation 4, we obtain

$$\pi_u b_u(\boldsymbol{x}_{t_s}) = p_{t_s,u}(t_s) = v_u(t_s) \cdot \epsilon.$$

If the path also includes the elements $(t_e - 1, j)$ and (t_e, i), then we have

$$a_{ji} \ b_i(\boldsymbol{x}_{t_e}) = p_{t_s,i}(t_e)/p_{t_s,j}(t_e - 1) = v_i(t_e) \cdot \epsilon / v_j(t_e - 1).$$

For the i-th state at time t_e,

$$p_{t_s,i}(t_e) = v_i(t_e) \cdot \epsilon^m.$$

Thus, we have

$$v_i(t_e) \geq \epsilon^{-\delta}.$$

From the second condition of Problem 1, it is obvious that the optimal path in the trellis structure gives the maximum cumulative likelihood in each group of subsequences. Thus, we obtain the two conditions of Lemma 1, which are equivalent to those of Problem 1. ■

Lemma 2: StreamScan guarantees the exactness of the result.

Proof: Let C_s be the starting position of the optimal subsequence $X[t_s : t_e]$. For *STS*, the optimal and overlapping subsequences start from the same position C_s since their paths share the same cell. If $s_i(t) \neq C_s$, the path of the subsequence does not overlap with the optimal path. Similarly, the upcoming subsequences do not overlap with the subsequence in the candidate subsequence set if

$$\forall_i, C_s \neq s_i(t).$$

StreamScan reports the subsequence $X[t_s : t_e]$ as the optimal subsequence only when the above condition is satisfied, which does not miss the optimal subsequence. Lemma 1 shows that the likelihood of the optimal subsequence can be computed from the cumulative likelihood. Thus, *StreamScan* guarantees the exactness of the result. ■

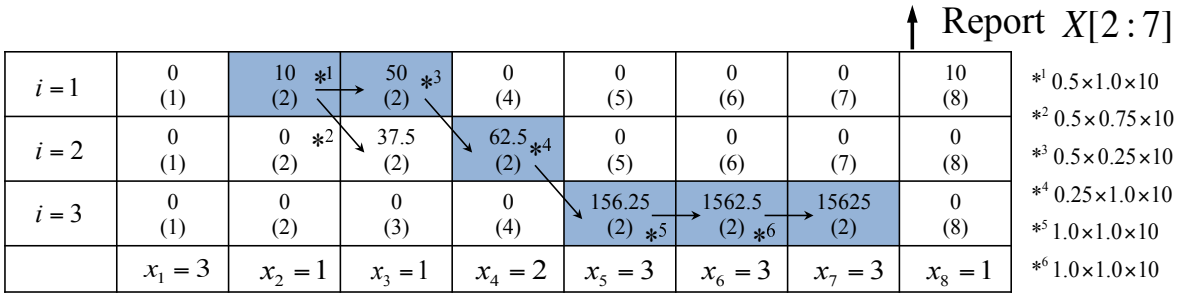


Fig. 5. Example of *StreamScan*. The upper number shows the cumulative likelihood value: $v_i(t)$, in each element of the score matrix. The number in parentheses shows the starting position: $s_i(t)$. The shaded elements denote the optimal Viterbi path. Starting from time-tick $t = 1$, it incrementally computes likelihood values $v_i(t)$ $(i = 1, 2, 3, t = 1, 2, \dots)$. At $t = 8$, we report $X[2 : 7]$ as the qualifying subsequence, since none of the upcoming subsequences will be the optimal subsequence.

B. Complexity

Let n be the length of an evolving sequence and k be the number of hidden states. We evaluate the computational complexity to maintain the trellis structure(s) in stream processing.

Lemma 3: SMM requires $O(n)$ and *SFR* needs $O(m)$ space and time per time-tick.

Proof: SMM has to maintain $O(n)$ trellis structures, and computes $O(nk^2)$ numbers every time-tick to identify qualifying subsequences. Thus, it requires $O(nk^2)$ time. Since this solution keeps two arrays of k numbers for each structure, overall, it needs $O(nk)$ space. Here, since k is a small constant value compared with n, the complexity can be simplified to $O(n)$. *SFR* needs to maintain $O(m)$ structures. For each structure, if the likelihood reaches the maximum, it stops the computation, where m is the length of the qualifying subsequences. ■

Lemma 4: StreamScan requires $O(1)$ space and time per time-tick.

Proof: StreamScan maintains a single trellis structure, and computes $O(k^2)$ numbers every time-tick. Thus, *StreamScan* requires $O(k^2)$ time per time-tick. *StreamScan* keeps two arrays of k numbers for the single structure, and it requires $O(k)$ space. Here, k is a constant value, thus the complexity is $O(1)$. ■

VI. Experiments

To evaluate the effectiveness of *StreamScan*, we carried out experiments on three real datasets. Our experiments were conducted on an Intel Core i7 2.5GHz with an 8GB memory. We performed this experiment with $k = 10$. We set δ at almost 10% of the sequence length for each dataset. The experiments were designed to answer the following questions:

1) How successful is *StreamScan* in capturing sequence patterns?
2) How does it scale with the sequence lengths n in terms of time and space?
3) How well does it handle diverse data streams?

A. Discovery of sequence patterns

We first demonstrate the effectiveness of our approach using real motion capture datasets. [3] It consists of sequences of 4-dimensional vectors (left/right legs and arms). Each sequence is a series of simple motions, such as walking and running. For each query model, we trained several basic motions, (such as "walking", "jumping", and "twisting"), using the Baum-Welch algorithm. We set $\epsilon = 10^{-10}$.

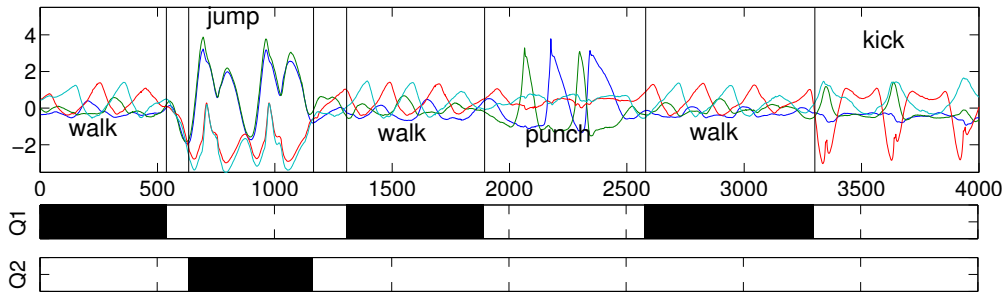
Our approach is useful for identifying human behavior. One of our results has already been presented in Figure 1. Similarly, Figure 6 shows how *StreamScan* detects the qualifying subsequences. It is robust against noise, and provides the correct segmentation and identifies all these specific motions. Note that *StreamScan* (and *SMM*, *SFR*) guarantees the exactness of the output, thus we omit the results of the other methods.

B. Scalability

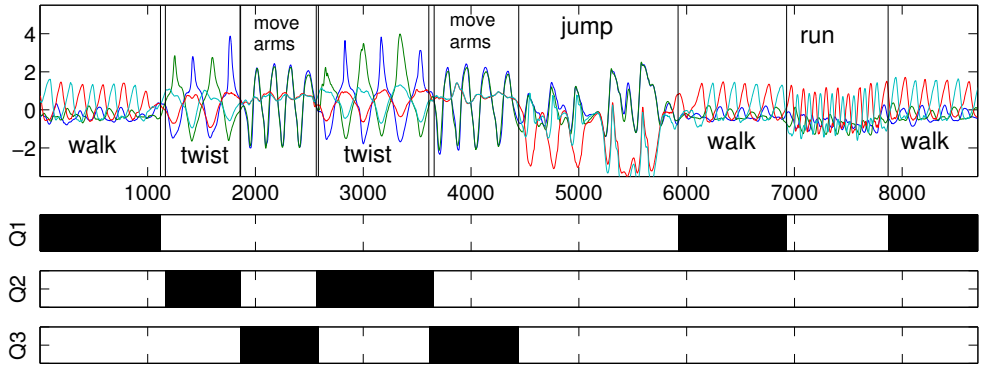
We performed experiments to evaluate the efficiency and to verify the complexity of *StreamScan*, which we discussed in Section V-B. Figure 7 compares *StreamScan* with two competitors, *SMM* and *SFR*, in terms of computation time for varying sequence lengths n. Figure 8 shows the amount of memory space required to maintain the trellis structure(s). The plots were generated using *MoCap*. In this dataset, the average length of qualifying subsequences is $m = 1,000 \sim 3,000$, thus we set $\epsilon = 10^{-10}$ and $\delta = 1,000$. The wall clock time is the average processing time needed to update the structure(s) for each time-tick.

As we expected, *StreamScan* identifies the qualifying subsequences much faster than the alternative solutions (See Figure 7). The trend shown in the figure agrees with our theoretical discussion in Section V-B. Compared with $O(n)$, which *SMM* requires, *StreamScan* achieves a dramatic reduction in computation time and space: it requires constant; i.e., it does not depend on n. Since *SFR* requires $O(m)$ time and space, our algorithm is up to 1,383 times faster than *SFR*. With respect to the memory space consumption, *SFR* needs an extra space to keep track of the candidate subsequence scores, and it depends on the length of captured data. However, the figure shows that the space requirement of *StreamScan* is clearly smaller than that of the alternative solutions.

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



(a) *MoCap* #2 (Query #1: walking, Query #2: jumping)



(b) *MoCap* #3 (Query #1: walking, Query #2: twisting, Query #3: moving arms)

Fig. 6. Discovery of sequence patterns in the *MoCap* dataset: Given several queries (i.e., (a) Q1:walking, Q2:jumping, (b) Q1:walking, Q2:twisting, Q3:moving arms), our algorithm perfectly identifies all sound parts from the streams (shown by the black rectangles).

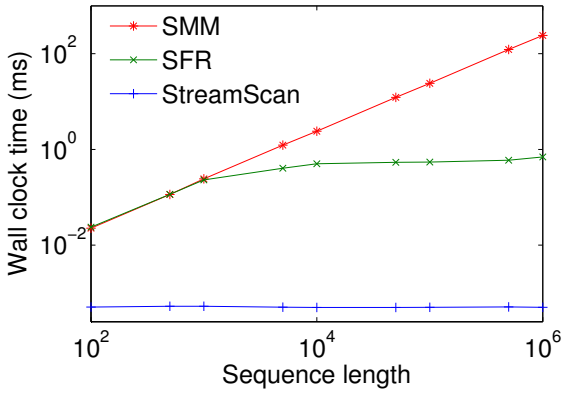


Fig. 7. Wall clock time vs. sequence length n. *StreamScan* is up to 479,000x faster than *SMM* and 1383x than *SFR*.

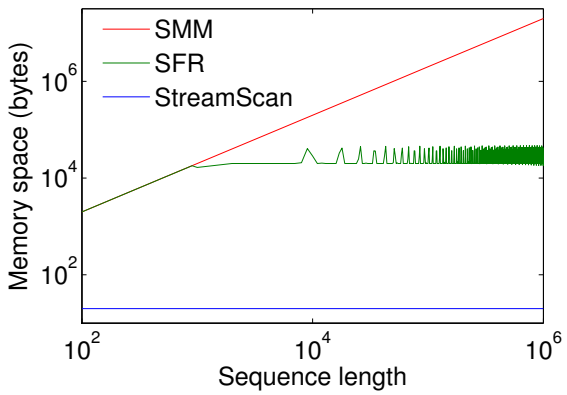


Fig. 8. Memory space consumption vs. sequence length n. *StreamScan* handles data streams with a small constant space.

C. StreamScan at work

StreamScan is capable of handling various types of data streams. Here, we describe some useful applications of our approach.

Social activity monitoring. One of our motivating applications is monitoring online social activity. Figure 9 shows the *WebClick* stream, which consists of 67GB web-click records of 2,582,252 users, obtained over one month (April, 2007). There are various types of URLs, such as "blog", and "news". Let us assume that we are given a query model of a specific activity pattern (e.g., a weekend pattern, learned from the first day in the stream). *StreamScan* can monitor the stream, and identify all weekend patterns according the given model. Also notice that it identifies one anomaly pattern at the end of the month (i.e., last Monday), and this is because of a national holiday (see the red circle).

Extreme detection in keyword stream. Figure 10 shows another natural and important application on *GoogleTrend* stream. This dataset consists of the volume of searches for

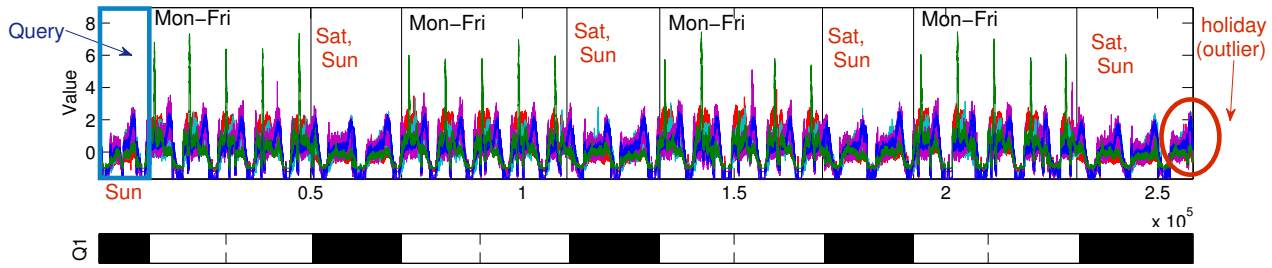


Fig. 9. Pattern discovery in the *WebClick* stream: given a stream of five time-evolving web-clicks (blog, news, dictionary, Q&A, mail site) over one month, every 10 sec., and a query model of a weekend pattern (learned with the first Sunday, in blue rectangle), *StreamScan* identifies all weekend patterns, with one exception (see the last day).

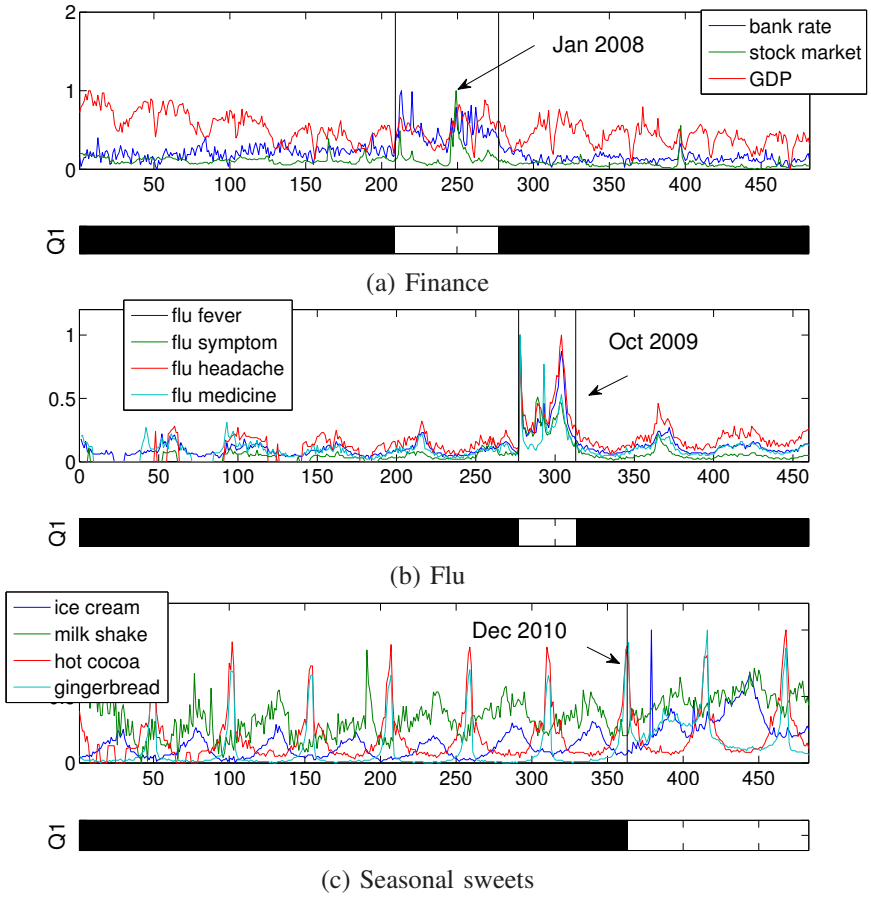


Fig. 10. Extreme detection in the *GoogleTrend* stream: the top figures show streams of co-evolving keywords, related to "finance", "flu" and "sweets" topics, covering nine years. The bottom figures show our discovery: we found several extreme patterns, (white rectangles), i.e., (a) global financial crisis in 2008, (b) swine flu outbreak in 2009, and (c) android OS ("Gingerbread", "Ice Cream Sandwich") released in 2010-2012.

various keywords on Google[4]. Each sequence represents search volumes related to keywords over time (over nine years, on a weekly basis). Consider that we are monitoring the web search activities. Our algorithm is applicable to finding "extreme" behavior in these streams. Specifically, Figure 10 shows the results of extreme monitoring with three different topics (e.g., finance, flu, and seasonal sweets). For each stream, we used the first three years (i.e., $X[1:156]$) as a query, learned model parameters, then monitored the stream. Here we report our discoveries.

[4]http://www.google.com/trends/

- (a) Finance: there is a yearly cyclic pattern, but there was a temporal change in January 2008, because of the global financial crisis.
- (b) Flu: there is a clear yearly periodicity; Starting every October, it slowly increases toward the peak in February. The only exception is October 2009, since that was when the swine flu pandemic spread around the world.
- (c) Seasonal sweets: each keyword has a yearly cycle, with a different phase; there are peaks in July for "ice

cream" and "milk shake", while there are peaks in December for "hot cocoa" and "gingerbread". However, the trend suddenly changed in Dec 2010. This is caused by the release of the android OS, called "Gingerbread", "Ice Cream Sandwich".

VII. Conclusions

We introduced the problem of subsequence identification with HMMs over data streams, and we proposed *StreamScan*, a new, fast algorithm to solve the problem. In conclusion, *StreamScan* has the following characteristics:

- It is **effective**: our experiments with real datasets show that *StreamScan* detects fruitful subsequences from diverse data streams.
- It is **exact**: it guarantees no false dismissals.
- It is **fast and nimble**: *StreamScan* requires only a single structure 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.

Acknowledgement

This work was supported by JSPS KAKENHI Grant-in-Aid for Scientific Research Number 24500138, 26730060, 26280112, Grant-in-Aid for JSPS Fellows Number 25·7946.

References

[1] D. Agarwal, B.-C. Chen, and P. Elango. Spatio-temporal models for estimating click-through rate. In *WWW Conference*, pages 21–30, Madrid, Spain, April 2009.

[2] G. E. Box, G. M. Jenkins, and G. C. Reinsel. *Time Series Analysis: Forecasting and Control*. Prentice Hall, Englewood Cliffs, NJ, 3rd edition, 1994.

[3] L. Chen and R. T. Ng. On the marriage of lp-norms and edit distance. In *VLDB*, pages 792–803, 2004.

[4] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *SIGMOD*, pages 419–429, 1994.

[5] Y. Fujiwara, Y. Sakurai, and M. Yamamuro. Spiral: efficient and exact model identification for hidden markov models. In *KDD*, pages 247–255, 2008.

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

[7] M. D. Hoffman, D. M. Blei, and F. R. Bach. Online learning for latent dirichlet allocation. In *NIPS*, pages 856–864, 2010.

[8] A. Jain, E. Y. Chang, and Y.-F. Wang. Adaptive stream resource management using kalman filters. In *SIGMOD*, pages 11–22, 2004.

[9] T. Kahveci and A. K. Singh. An efficient index structure for string databases. In *Proceedings of VLDB*, pages 351–360, September 2001.

[10] E. J. Keogh, S. Chu, D. Hart, and M. J. Pazzani. An online algorithm for segmenting time series. In *ICDM*, pages 289–296, 2001.

[11] E. J. Keogh, T. Palpanas, V. B. Zordan, D. Gunopulos, and M. Cardle. Indexing large human-motion databases. In *VLDB*, pages 780–791, 2004.

[12] J. Letchner, C. Ré, M. Balazinska, and M. Philipose. Access methods for markovian streams. In *ICDE*, pages 246–257, 2009.

[13] L. Li, C.-J. M. Liang, J. Liu, S. Nath, A. Terzis, and C. Faloutsos. Thermocast: A cyber-physical forecasting model for data centers. In *KDD*, 2011.

[14] L. Li, J. McCann, N. Pollard, and C. Faloutsos. Dynammo: Mining and summarization of coevolving sequences with missing values. In *KDD*, 2009.

[15] L. Li and B. A. Prakash. Time series clustering: Complex is simpler! In *ICML*, 2011.

[16] J. Lin, E. J. Keogh, S. Lonardi, J. P. Lankford, and D. M. Nystrom. Visually mining and monitoring massive time series. In *KDD*, pages 460–469, 2004.

[17] M. Mathioudakis, N. Koudas, and P. Marbach. Early online identification of attention gathering items in social media. In *WSDM*, pages 301–310, 2010.

[18] Y. Matsubara, L. Li, E. E. Papalexakis, D. Lo, Y. Sakurai, and C. Faloutsos. F-trail: Finding patterns in taxi trajectories. In *PAKDD (1)*, pages 86–98, 2013.

[19] Y. Matsubara, Y. Sakurai, and C. Faloutsos. Autoplait: Automatic mining of co-evolving time sequences. In *SIGMOD*, 2014.

[20] Y. Matsubara, Y. Sakurai, C. Faloutsos, T. Iwata, and M. Yoshikawa. Fast mining and forecasting of complex time-stamped events. In *KDD*, pages 271–279, 2012.

[21] Y. Matsubara, Y. Sakurai, B. A. Prakash, L. Li, and C. Faloutsos. Rise and fall patterns of information diffusion: model and implications. In *KDD*, pages 6–14, 2012.

[22] Y. Matsubara, Y. Sakurai, and M. Yoshikawa. Scalable algorithms for distribution search. In *ICDM*, pages 347–356, 2009.

[23] A. Mueen and E. J. Keogh. Online discovery and maintenance of time series motifs. In *KDD*, pages 1089–1098, 2010.

[24] 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.

[25] S. Papadimitriou and P. S. Yu. Optimal multi-scale patterns in time series streams. In *SIGMOD Conference*, pages 647–658, 2006.

[26] P. Papapetrou, V. Athitsos, M. Potamias, G. Kollios, and D. Gunopulos. Embedding-based subsequence matching in time-series databases. *ACM Trans. Database Syst.*, 36(3):17, 2011.

[27] P. Patel, E. J. Keogh, J. Lin, and S. Lonardi. Mining motifs in massive time series databases. In *Proceedings of ICDM*, pages 370–377, 2002.

[28] T. Rakthanmanon, B. J. L. Campana, A. Mueen, G. E. A. P. A. Batista, M. B. Westover, Q. Zhu, J. Zakaria, and E. J. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *KDD*, pages 262–270, 2012.

[29] Y. Sakurai, C. Faloutsos, and M. Yamamuro. Stream monitoring under the time warping distance. In *ICDE*, pages 1046–1055, Istanbul, Turkey, April 2007.

[30] Y. Sakurai, S. Papadimitriou, and C. Faloutsos. Braid: Stream mining through group lag correlations. In *SIGMOD*, pages 599–610, 2005.

[31] M.-C. Silaghi. Spotting subsequences matching an hmm using the average observation probability criteria with application to keyword spotting. In *AAAI*, pages 1118–1123, 2005.

[32] Y. Tao, C. Faloutsos, D. Papadias, and B. Liu. Prediction and indexing of moving objects with unknown motion patterns. In *SIGMOD*, pages 611–622, 2004.

[33] M. Vlachos, G. Kollios, and D. Gunopulos. Elastic translation invariant matching of trajectories. *Mach. Learn.*, 58(2-3):301–334, Feb. 2005.

[34] M. Vlachos, S. S. Kozat, and P. S. Yu. Optimal distance bounds on time-series data. In *SDM*, pages 109–120, 2009.

[35] P. Wang, H. Wang, and W. Wang. Finding semantics in time series. In *SIGMOD Conference*, pages 385–396, 2011.

[36] J. G. Wilpon, C. H. Lee, and L. R. Rabiner. Application of hidden Markov models for recognition of a limited set of words in unconstrained speech. In *ICASSP*, pages 254–257 vol.1, 1989.

[37] J. G. Wilpon, L. R. Rabiner, C. H. Lee, and E. R. Goldman. Automatic recognition of keywords in unconstrained speech using hidden Markov models. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 38(11):1870–1878, 1990.