Efficient Episode Mining of Dynamic Event Streams

Debprakash Patnaik*, Srivatsan Laxman†, Badrish Chandramouli‡ and Naren Ramakrishnan§
*Amazon.com, Seattle, WA 98109, USA; E-mail: patnaikd@amazon.com
†Microsoft Research, Bangalore, India 560080; E-mail: slaxman@microsoft.com
‡Microsoft Research, Redmond, WA 98052; E-mail: badrishc@microsoft.com
§Department of Computer Science, Virginia Tech, Blacksburg, VA 24061; E-mail: naren@vt.edu

***Abstract*—Discovering frequent episodes over event sequences is an important data mining problem. Existing methods typically require multiple passes over the data, rendering them unsuitable for streaming contexts. We present the first streaming algorithm for mining frequent episodes over a window of recent events in the stream. We derive approximation guarantees for our algorithm in terms of: (i) the separation of frequent episodes from infrequent ones, and (ii) the rate of change of stream characteristics. Our parameterization of the problem provides a new sweet spot in the tradeoff between making distributional assumptions over the stream and algorithmic efficiencies of mining. We illustrate how this yields significant benefits when mining practical streams from neuroscience and telecommunications logs.**

***Index Terms*—Event Sequences; Data Streams; Frequent Episodes; Pattern Discovery; Streaming Algorithms; Approximation Algorithms**

I. Introduction

Application contexts in telecommunications, neuroscience, and intelligence analysis feature massive data streams [1] with 'firehose'-like rates of arrival. In many cases, we need to analyze such streams at speeds comparable to their generation rate. In neuroscience, one goal is to track spike trains from multi-electrode arrays [2] with a view to identify cascading circuits of neuronal firing patterns. In telecommunications, network traffic and call logs must be analyzed on a continual basis to detect attacks or other malicious activity. The common theme in all these scenarios is the need to mine episodes (i.e., a succession of events occurring frequently, but not necessarily consecutively [3]) from dynamic and evolving streams.

Algorithms for pattern mining over streams have become increasingly popular over the recent past [4]–[7]. Manku and Motwani [4] introduced a lossy counting algorithm for approximate frequency counting over streams, with no assumptions on the stream. Their focus on a worst-case setting often leads to stringent threshold requirements. At the other extreme, algorithms such as [5] provide significant efficiencies in mining but make strong assumptions such as i.i.d distribution of symbols in a stream.

In the course of analyzing some real-world datasets, we were motivated to develop new methods as existing methods are unable to process streams at the rate and quality guarantees desired (see Sec. VI for some examples). Furthermore, established stream mining algorithms are almost entirely focused on itemset mining (and, modulo a few isolated exceptions, just the counting phase of it) whereas we are interested in mining general episodes.

Our specific contributions are as follows:

- We present the *first* algorithm for mining episodes in a stream. Unlike prior streaming algorithms that focus almost exclusively on counting, we provide solutions for both candidate generation and counting over a stream.
- Devoid of any statistical assumptions on the stream (e.g., independence or otherwise), we develop a novel error characterization for streaming episodes by identifying and tracking two key properties of the stream, viz. *maximum rate of change* and *top-k separation*. We demonstrate how the use of these two properties enables novel algorithmic optimizations, such as the idea of *borders* to amortize work as the stream is tracked.
- Although our work is geared towards episode mining, we adopt a black-box model of an episode mining algorithm. In other words, our approach can encapsulate and wrap around any pattern discovery algorithm to enable it to accommodate streaming data. This significantly generalizes the scope and applicability of our approach as a general methodology to *streamify* existing pattern discovery algorithms.
- We demonstrate successful applications in neuroscience and telecommunications log analysis, and illustrate significant benefits in runtime, memory usage, and the scales of data that can be mined. We compare against episode-mining adaptations of two typical algorithms [5] from streaming itemsets literature.

II. Preliminaries

In the framework of frequent episodes [3], an *event sequence* is denoted as $\langle (e_1, \tau_1), \ldots, (e_n, \tau_n) \rangle$, where (e_i, τ_i) represents the i^{th} event; e_i is drawn from a finite alphabet $\mathcal{E}$ of symbols (called *event-types*) and τ_i denotes the time-stamp of the i^{th} event, with $\tau_{i+1} \geq \tau_i$, $i = 1, \ldots, (n-1)$. An ℓ-*node episode* α is defined by a triple $\alpha = (V_\alpha, <_\alpha, g_\alpha)$, where $V_\alpha = \{v_1, \ldots, v_\ell\}$ is a collection of ℓ nodes, $<_\alpha$ is a partial order over V_α and $g_\alpha : V_\alpha \rightarrow \mathcal{E}$ is a map that assigns an event-type $g_\alpha(v)$ to each node $v \in V_\alpha$. An *occurrence* of an episode α is a map $h : V_\alpha \rightarrow \{1, \ldots, n\}$ such that $e_{h(v)} = g_\alpha(v)$ for all $v \in V_\alpha$ and for all pairs of nodes $v, v' \in V_\alpha$ such that $v <_\alpha v'$ the map h ensures that $\tau_{h(v)} < \tau_{h(v')}$. Two occurrences of an episode are *non-overlapped* [8] if no event corresponding to one appears in-between the events corresponding to the other. The maximum number of non-overlapped occurrences of an episode is defined as its *frequency* in the event sequence.

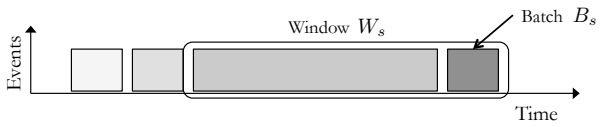


Fig. 1. A sliding window model for episode mining over event streams: B_s is the most recent batch of events that arrived in the stream and W_s is the window of interest over which the user wants to determine the set of frequent episodes.

The task in frequent episode discovery is to find all episodes whose frequency exceeds a user-defined threshold. Apriori-style level-wise algorithms [3], [8] are typically applicable in this setting. An important variant is top-k episode mining (see [9] for definitions in the itemsets mining context), where, rather than a frequency threshold, the user supplies the *number* of most frequent episodes needed.

Definition 1 (Top-k episodes of size ℓ): The set of top-k episodes of size ℓ is defined as the collection of all ℓ-node episodes with frequency *greater than or equal to* the frequency f^k of the k^{th} most frequent ℓ-node episode in the given event sequence.

The number of top-k ℓ-node episodes can exceed k, although the number of ℓ-node episodes with frequencies strictly greater than f^k is at most $(k-1)$. In general, top-k mining can be difficult to solve without knowledge of a good lower-bound for f^k; for relatively short event sequences the following simple solution works well-enough: start mining at a high threshold and progressively lower the threshold until the desired number of top patterns are returned.

III. Problem Statement

The data available (referred to as an *event stream*) is in the form of a potentially infinite sequence of events:

$$\mathcal{D} = \langle (e_1, \tau_1), (e_2, \tau_2), \ldots, (e_i, \tau_i), \ldots, (e_n, \tau_n), \ldots \rangle \quad (1)$$

Our goal is to find all episodes that were frequent in the recent past; for this, we consider a sliding window model[1] for the *window of interest*. In this model, the user wants to determine episodes that were frequent over a (historical) window of fixed-size terminating at the current time-tick. As new events arrive in the stream, the user's window of interest shifts, and the data mining task is to next report the frequent episodes in the new window of interest.

We consider the case where the window of interest is very large and cannot be stored and processed in-memory. This straightaway precludes the use of standard multi-pass algorithms for frequent episode discovery over the window of interest. We organize the events in the stream into smaller batches such that at any given time only the latest incoming batch is stored and processed in memory. This is illustrated in Fig. 1. The current window of interest is denoted by W_s and the most recent batch, B_s, consists of events in $\mathcal{D}$ that occurred between times $(s-1)T_b$ and sT_b where T_b is the time-span of each batch and s is the batch number ($s = 1, 2, \ldots$) .

[1]Other models such as the landmark and time-fading models have also been studied [7] but we do not consider them here.

Pattern	Count
WXYZ	12
PQRS	10
ABCD	8
MNOP	8
EFGH	0
IJKL	0

B_1

Pattern	Count
EFGH	15
IJKL	12
MNOP	10
ABCD	9
PQRS	0
WXYZ	0

B_2

Pattern	Count
WXYZ	12
EFGH	10
ABCD	10
MNOP	8
PQRS	0
IJKL	0

B_3

Pattern	Count
IJKL	11
PQRS	9
MNOP	8
ABCD	8
EFGH	0
WXYZ	0

B_4

Fig. 2. Batch frequencies in *Example 1*.

TABLE I
WINDOW FREQUENCIES IN *Example 1*.

Episode	ABCD	MNOP	EFGH	WXYZ	IJKL	PQRS
Window Freq	35	34	25	24	23	19

The frequency of an episode α in a batch B_s is referred to as its *batch frequency* $f^s(\alpha)$. The current *window of interest*, W_s, consists of m consecutive batches ending in batch B_s, i.e.

$$W_s = \langle B_{s-m+1}, B_{s-m+2}, \ldots, B_s \rangle \quad (2)$$

Definition 2 (Window Frequency): The frequency of an episode α over window W_s, referred to as its *window frequency* and denoted by $f^{W_s}(\alpha)$, is defined as the sum of batch frequencies of α in W_s. Thus, if $f^j(\alpha)$ denotes the batch frequency of α in batch B_j, then the window frequency of α is given by $f^{W_s}(\alpha) = \sum_{B_j \in W_s} f^j(\alpha)$.

In summary, we are given an event stream ($\mathcal{D}$), a time-span for batches (T_b), the number of consecutive batches that constitute the current window of interest (m), the desired size of frequent episodes (ℓ), the desired number of most frequent episodes (k) and the problem is to discover the top-k episodes in the current window without actually having the entire window in memory.

Problem 1 (Streaming Top-k Mining): For each new batch, B_s, of events in the stream, find all ℓ-node episodes in the corresponding window of interest, W_s, whose window frequencies are greater than or equal to the window frequency, f_s^k, of k^{th} most frequent ℓ-node episode in W_s.

Example 1 (Window Top-k v/s Batch Top-k): Let W be a window of four batches B_1 through B_4. The episodes in each batch with corresponding batch frequencies are listed in Fig. 2. The corresponding window frequencies (sum of each episodes' batch frequencies) are listed in Table I. The top-2 episodes in B_1 are (PQRS) and (WXYZ). Similarly (EFGH) and (IJKL) are the top-2 episodes in B_2, and so on. (ABCD) and (MNOP) have the highest window frequencies but never appear in the top-2 of any batch – these episodes would 'fly below the radar' and go undetected if we considered only the top-2 episodes in every batch as candidates for the top-2 episodes over W. This example can be easily generalized to any number of batches and any k.

Example 1 highlights the main challenge in the streaming top-k mining problem: we can only store/process the most recent batch of events in the window of interest and the batchwise top-k may not contain sufficient information to compute the top-k over the entire window. It is obviously not possible to track all episodes (both frequent and infrequent) in every batch since the pattern space is typically very large. This brings us to the question of which episodes to track in

every batch – how deep must we search within each batch for episodes that have potential to become top-k over the window? We develop the formalism needed to answer this question.

IV. Persistence and Top-k Approximation

We identify two important properties of the underlying event stream which influence the design and analysis of our algorithms. These are stated in *Definitions 3 & 4* below.

Definition 3 (Maximum Rate of Change, Δ): Maximum rate of change $\Delta(> 0)$ is defined as the maximum change in batch frequency of any episode, α, across any pair of consecutive batches, B_s and B_{s+1}, i.e., $\forall \alpha, s$, we have

$$|f^{s+1}(\alpha) - f^s(\alpha)| \leq \Delta. \tag{3}$$

Intuitively, Δ controls the extent of change from one batch to the next. While it is trivially bounded by the number of events arriving per batch, it is often much smaller in-practice.

Definition 4 (Top-k Separation of (φ, ϵ)): A batch B_s of events is said to have a *top-k separation of* (φ, ϵ), $\varphi \geq 0$, $\epsilon \geq 0$, if it contains at most $(1+\epsilon)k$ episodes with batch frequencies of $(f_k^s - \varphi\Delta)$ or more, where f_k^s is the batch frequency of the k^{th} most-frequent episode in B_s and Δ is the maximum rate of change.

This is essentially a measure of how well-separated the frequencies of the top-k episodes are relative to the rest of the episodes. We expect to see roughly k episodes with batch frequencies of at least f_s^k and the separation is considered to be high (or good) if lowering the threshold from f_s^k to $(f_k^s - \varphi\Delta)$ only brings-in very few additional episodes, i.e. ϵ remains small as φ increases. Top-k separation of any batch B_s is characterized by, not one but, several pairs of (φ, ϵ) since φ and ϵ are functionally related: ϵ is typically close to zero if $\varphi = 0$, while we have ϵk roughly the size of the class of ℓ-size episodes (minus k) if $\varphi\Delta \geq f_s^k$. Note that ϵ is a non-decreasing function of φ and that top-k separation is measured relative to the maximum rate of change Δ.

We now use the maximum rate of change property to design efficient streaming algorithms for top-k episode mining and show that top-k separation plays a pivotal role in determining the quality of approximation that our algorithms achieve.

Lemma 1: The batch frequencies of the k^{th} most-frequent episodes in any pair of consecutive batches cannot differ by more than the maximum rate of change Δ, i.e., for every batch B_s, we must have

$$|f_k^{s+1} - f_k^s| \leq \Delta. \tag{4}$$

The above lemma follows directly from: (i) there are at least k episodes with frequencies no less than f_k^s, and (ii) the batch frequency of any episode can increase or decrease by no more than Δ when going from one batch to the next.

Our next observation is that if the batch frequency of an episode is known relative to f_k^s in the current batch B_s, we can bound its frequency in any later batch B_{s+r}.

Lemma 2: Consider two batches, B_s and B_{s+r}, $r \in \mathbb{Z}$, located r batches away from each other. Under a maximum rate of change of Δ the batch frequency of any episode α in B_{s+r} must satisfy the following:

1) If $f^s(\alpha) \geq f_k^s$, then $f^{s+r}(\alpha) \geq f_k^{s+r} - 2|r|\Delta$
2) If $f^s(\alpha) < f_k^s$, then $f^{s+r}(\alpha) < f_k^{s+r} + 2|r|\Delta$

Detailed proofs can be found in [10]. *Lemma 2* gives us a way to track episodes that have potential to be in the top-k of future batches. This is an important property which our algorithm exploits and we recorded this as a remark below.

Remark 1: The top-k episodes of batch, B_{s+r}, $r \in \mathbb{Z}$, must have batch frequencies of at least $(f_k^s - 2|r|\Delta)$ in batch B_s. Specifically, the top-k episodes of B_{s+1} must have batch frequencies of at least $(f_k^s - 2\Delta)$ in B_s.

The maximum rate of change property leads to a necessary condition, in the form of a minimum batch-wise frequency, for an episode α to be in the top-k over a window W_s.

Theorem 1 (Exact Top-k over W_s): An episode, α, can be a top-k episode over window W_s only if its batch frequencies satisfy $f^{s'}(\alpha) \geq (f_k^{s'} - 2(m-1)\Delta)\ \forall B_{s'} \in W_s$.

Proof: Consider an episode β for which $f^{s'}(\beta) < (f_k^{s'} - 2(m-1)\Delta)$ in batch $B_{s'} \in W_s$. Let α be any top-k episode of $B_{s'}$. In any other batch $B_p \in W_s$, we have

$$\begin{aligned} f^p(\alpha) &\geq f^{s'}(\alpha) - |p - s'|\Delta \\ &\geq f_k^{s'} - |p - s'|\Delta \end{aligned} \tag{5}$$

and

$$\begin{aligned} f^p(\beta) &\leq f^{s'}(\beta) + |p - s'|\Delta \\ &< (f_k^{s'} - 2(m-1)\Delta) + |p - s'|\Delta \end{aligned} \tag{6}$$

Applying $|p - s'| \leq (m-1)$ to the above, we get

$$f^p(\alpha) \geq f_k^{s'} - (m-1)\Delta > f^p(\beta) \tag{7}$$

This implies $f^{W_s}(\beta) < f^{W_s}(\alpha)$ for every top-k episode α of $B_{s'}$. Since there are at least k top-k episodes in $B_{s'}$, β cannot be a top-k episode over the window W_s. ∎

Based on *Theorem 1* we have the following simple algorithm for obtaining the top-k episodes over a window: Use a traditional level-wise approach to find all episodes with a batch frequency of at least $(f_1^k - 2(m-1)\Delta)$ in the first batch (B_1), accumulate their corresponding batch frequencies over all m batches of W_s and report the episodes with the k highest window frequencies over W_s. This approach is guaranteed to return the exact top-k episodes over W_s. In order to report the top-k over the next sliding window W_{s+1}, we need to consider all episodes with batch frequency of at least $(f_2^k - 2(m-1)\Delta)$ in the second batch and track them over all batches of W_{s+1}, and so on. Thus, an exact solution to *Problem 1* would require running a level-wise episode mining algorithm in every batch, B_s, $s = 1, 2, \ldots$, with a frequency threshold of $(f_s^k - 2(m-1)\Delta)$.

A. Class of (v, k)-Persistent Episodes

Theorem 1 characterizes the minimum batchwise computation needed in order to obtain the exact top-k episodes over a sliding window. This is effective when Δ and m are small (compared to f_s^k). However, the batchwise frequency

thresholds can become very low in other settings, making the processing time per-batch as well as the number of episodes to track over the window to become impractically high. To address this issue, we introduce a new class of episodes called (v,k)-*persistent episodes* which can be computed efficiently by employing higher batchwise thresholds. Further, we show that these episodes can be used to approximate the true top-k episodes over the window and the quality of approximation is characterized in terms of the top-k separation property (cf. *Definition 4*).

Definition 5 ((v,k)-Persistent Episode): An episode is said to be (v,k)-*persistent* over window W_s if it is a top-k episode in *at least* v batches of W_s.

Problem 2 (Mining (v,k)-Persistent Episodes): For each new batch, B_s, of events in the stream, find all ℓ-node (v,k)-persistent episodes in the corresponding window of interest, W_s.

Theorem 2: An episode, α, can be (v,k)-persistent over the window W_s only if its batch frequencies satisfy $f^{s'}(\alpha) \geq (f_k^{s'} - 2(m-v)\Delta)$ for every batch $B_{s'} \in W_s$.

Proof: Let α be (v,k)-persistent over W_s and let V_α denote the set of batches in W_s in which α is in the top-k. For any $B_q \notin V_\alpha$ there exists $B_{\widehat{p}(q)} \in V_\alpha$ that is *nearest to* B_q. Since $|V_\alpha| \geq v$, we must have $|\widehat{p}(q) - q| \leq (m-v)$. Applying *Lemma 2* we then get $f^q(\alpha) \geq f_k^q - 2(m-v)\Delta$ for all $B_q \notin V_\alpha$. ∎

Theorem 2 gives us the necessary conditions for computing all (v,k)-persistent episodes over sliding windows in the stream. The batchwise threshold required for (v,k)-persistent episodes depends on the parameter v. For $v=1$, the threshold coincides with the threshold for exact top-k in *Theorem 1*. The threshold increases linearly with v and is highest at $v=m$ (when the batchwise threshold is same as the corresponding batchwise top-k frequency).

The algorithm for discovering (v,k)-persistent episodes follows the same general lines as the one described earlier for exact top-k mining, only that we now apply higher batchwise thresholds: For each new batch, B_s, entering the stream, use a standard level-wise episode mining algorithm to find all episodes with batch frequency of at least $(f_s^k - 2(m-v)\Delta)$. We provide more details of our algorithm later in Sec. V. First, we investigate the quality of approximation of top-k that (v,k)-persistent episodes offer and show that the number of errors is closely related to the degree of top-k separation.

1) Top-k Approximation: The main idea here is that, under a maximum rate of change Δ and a top-k separation of (φ,ϵ), there cannot be too many distinct episodes which are not (v,k)-persistent while still having sufficiently high window frequencies. To this end, we first compute a lower-bound (f_L) on the window frequencies of (v,k)-persistent episodes and an upper-bound (f_U) on the window frequencies of episodes that are *not* (v,k)-persistent (cf. *Lemmas 3 & 4*).

Lemma 3: If episode α is (v,k)-persistent over a window, W_s, then its window frequency, $f^{W_s}(\alpha)$, must satisfy the following lower-bound:

$$f^{W_s}(\alpha) \geq \sum_{B_{s'}} f_k^{s'} - (m-v)(m-v+1)\Delta \stackrel{\text{def}}{=} f_L \qquad (8)$$

Proof: Consider episode α that is (v,k)-persistent over W_s and let V_α denote the batches of W_s in which α is in the top-k. The window frequency of α can be written as

$$\begin{aligned} f^{W_s}(\alpha) &= \sum_{B_p \in V_\alpha} f^p(\alpha) + \sum_{B_q \in W_s \setminus V_\alpha} f^q(\alpha) \\ &\geq \sum_{B_p \in V_\alpha} f_k^p + \sum_{B_q \in W_s \setminus V_\alpha} f_k^q - 2|\widehat{p}(q) - q|\Delta \\ &= \sum_{B_{s'} \in W_s} f_k^{s'} - \sum_{B_q \in W_s \setminus V_\alpha} 2|\widehat{p}(q) - q|\Delta \qquad (9) \end{aligned}$$

where $B_{\widehat{p}(q)} \in V_\alpha$ denotes the batch nearest B_q where α is in the top-k. Since $|W_s \setminus V_\alpha| \leq (m-v)$, we must have

$$\begin{aligned} \sum_{B_q \in W_s \setminus V_\alpha} |\widehat{p}(q) - q| &\leq (1 + 2 + \cdots + (m-v)) \\ &= \frac{1}{2}(m-v)(m-v+1) \qquad (10) \end{aligned}$$

Putting together (9) and (10) gives us the lemma. ∎

Similar arguments give us the next lemma about the maximum frequency of episodes that are *not* (v,k)-persistent (Full proofs are available in [10]).

Lemma 4: If episode β is not (v,k)-persistent over a window, W_s, then its window frequency, $f^{W_s}(\beta)$, must satisfy the following upper-bound:

$$f^{W_s}(\beta) < \sum_{B_{s'}} f_k^{s'} + v(v+1)\Delta \stackrel{\text{def}}{=} f_U \qquad (11)$$

It turns out that $f_U > f_L \ \forall v,\ 1 \leq v \leq m$, and hence there is always a possibility for some episodes which are not (v,k)-persistent to end up with higher window frequencies than one or more (v,k)-persistent episodes. We observed a specific instance of this kind of 'mixing' in our motivating example as well (cf. *Example 1*). This brings us to the top-k separation property that we introduced in *Definition 4*. Intuitively, if there is sufficient separation of the top-k episodes from the rest of the episodes in every batch, then we would expect to see very little mixing. As we shall see, this separation need not occur exactly at k^{th} most-frequent episode in every batch, somewhere close to it is sufficient to achieve a good top-k approximation.

Definition 6 (Band Gap Episodes, $\mathcal{G}_\varphi$): In any batch $B_{s'} \in W_s$, the half-open frequency interval $[f_{s'}^k - \varphi\Delta,\ f_{s'}^k)$ is called the *band gap* of $B_{s'}$. The corresponding set, $\mathcal{G}_\varphi$, of *band gap episodes* over the window W_s, is defined as the collection of all episodes with batch frequencies in the band gap of at least one $B_{s'} \in W_s$.

The main feature of $\mathcal{G}_\varphi$ is that, if φ is large-enough, then the only episodes which are not (v,k)-persistent but that can still mix with (v,k)-persistent episodes are those belonging

to $\mathcal{G}_\varphi$. This is stated formally in the next lemma. The proof, omitted here, can be found in [10].

Lemma 5: If $\frac{\varphi}{2} > \max\{1, (1-\frac{v}{m})(m-v+1)\}$, then any episode β that is *not* (v,k)-persistent over W_s, can have $f^{W_s}(\beta) \geq f_L$ only if $\beta \in \mathcal{G}_\varphi$.

The number of episodes in $\mathcal{G}_\varphi$ is controlled by the top-k separation property, and since many of the non-persistent episodes which can mix with persistent ones must spend not one, but several batches in the band gap, the number of unique episodes that can cause such errors is bounded. *Theorem 3* is our main result about quality of top-k approximation that (v,k)-persistence can achieve.

Theorem 3 (Quality of Top-k Approximation): Let every batch $B_{s'} \in W_s$ have a top-k separation of (φ, ϵ) with $\frac{\varphi}{2} > \max\{1, (1-\frac{v}{m})(m-v+1)\}$. Let $\mathcal{P}$ denote the set of all (v,k)-persistent episodes over W_s. If $|\mathcal{P}| \geq k$, then the top-k episodes over W_s can be determined from $\mathcal{P}$ with an error of no more than $\left(\frac{\epsilon k m}{\mu}\right)$ episodes, where $\mu = \min\{m-v+1, \frac{\varphi}{2}, \frac{1}{2}(\sqrt{1+2m\varphi}-1)\}$.

Proof: By top-k separation, we have a maximum of $(1+\epsilon)k$ episodes in any batch $B_{s'} \in W_s$, with batch frequencies greater than or equal to $f^k_{s'} - \varphi\Delta$. Since at least k of these must belong to the top-k of the $B_{s'}$, there are no more than ϵk episodes that can belong to the band gap of $B_{s'}$. Thus, there can be no more than a total of $\epsilon k m$ episodes over all m batches of W_s that can belong to $\mathcal{G}_\varphi$.

Consider any $\beta \notin \mathcal{P}$ with $f^{W_s}(\beta) \geq f_L$ – these are the only episodes whose window frequencies can exceed that of any $\alpha \in \mathcal{P}$ (since f_L is the minimum window frequency of any α). If μ denotes the minimum number of batches in which β belongs to the band gap, then there can be at most $\left(\frac{\epsilon k m}{\mu}\right)$ such *distinct* β. Thus, if $|\mathcal{P}| \geq k$, we can determine the set of top-k episodes over W_s with error no more than $\left(\frac{\epsilon k m}{\mu}\right)$ episodes.

There are now two cases to consider to determine μ: (i) β is in the top-k of some batch, and (ii) β is not in the top-k of any batch.

Case (i): Let β be in the top-k of $B_{s'} \in W_s$. Let $B_{s''} \in W_s$ be t batches away from $B_{s'}$. Using *Lemma 2* we get $f^{s''}(\beta) \geq f^k_{s''} - 2t\Delta$. The minimum t for which $(f^k_{s''} - 2t\Delta < f^k_s - \varphi\Delta)$ is $\left(\frac{\varphi}{2}\right)$. Since $\beta \notin \mathcal{P}$, β is below the top-k in at least $(m-v+1)$ batches. Hence β stays in the band gap of at least $\min\{m-v+1, \frac{\varphi}{2}\}$ batches of W_s.

Case (ii): Let V_G denote the set of batches in W_s where β lies in the band gap and let $|V_G| = g$. Since β does not belong to top-k of any batch, it must stay below the band gap in all the $(m-g)$ batches of $(W_s \setminus V_G)$. Since Δ is the maximum rate of change, the window frequency of β can be written as follows:

$$\begin{aligned} f^{W_s}(\beta) &= \sum_{B_p \in V_G} f^p(\beta) + \sum_{B_q \in W_s \setminus V_G} f^q(\beta) \\ &< \sum_{B_p \in V_G} f^p(\beta) + \sum_{B_q \in W_s \setminus V_G} (f^k_q - \varphi\Delta) \quad (12) \end{aligned}$$

Let $B_{\widehat{q}(p)}$ denote the batch in $W_s \setminus V_G$ that is nearest to $B_p \in V_G$. Then we have:

$$\begin{aligned} f^p(\beta) &\leq f^{\widehat{q}(p)}(\beta) + |p - \widehat{q}(p)|\Delta \\ &< f^k_{\widehat{q}(p)} - \varphi\Delta + |p - \widehat{q}(p)|\Delta \\ &< f^k_p - \varphi\Delta + 2|p - \widehat{q}(p)|\Delta \quad (13) \end{aligned}$$

where the second inequality holds because β is below the band gap in $B_{\widehat{q}(p)}$ and (13) follows from *Lemma 1*. Using (13) in (12) we get

$$\begin{aligned} f^{W_s}(\beta) &< \sum_{B_{s'} \in W_s} f^k_{s'} - m\varphi\Delta + \sum_{B_p \in V_G} 2|p - \widehat{q}(p)|\Delta \\ &< \sum_{B_{s'} \in W_s} f^k_{s'} - m\varphi\Delta + 2(1+2+\cdots+g)\Delta \\ &= \sum_{B_{s'} \in W_s} f^k_{s'} - m\varphi\Delta + g(g+1)\Delta = \text{UB} \quad (14) \end{aligned}$$

The smallest g for which $(f^{W_s}(\beta) \geq f_L)$ is feasible can be obtained by setting $\text{UB} \geq f_L$. Since $\frac{\varphi}{2} > (1-\frac{v}{m})(m-v+1)$, $\text{UB} \geq f_L$ implies

$$\sum_{B_{s'} \in W_s} f^k_{s'} - m\varphi\Delta + g(g+1)\Delta > \sum_{B_{s'} \in W_s} f^k_{s'} - \frac{m\varphi\Delta}{2}$$

Solving for g, we get $g \geq \frac{1}{2}(\sqrt{1+2m\varphi}-1)$. Combining cases (i) and (ii), we get $\mu = \min\{m-v+1, \frac{\varphi}{2}, \frac{1}{2}(\sqrt{1+2m\varphi}-1)\}$. ■

Theorem 3 shows the relationship between the extent of top-k separation required and quality of top-k approximation that can be obtained through (v,k)-persistent episodes. In general, μ (which is minimum of three factors) increases with $\frac{\varphi}{2}$ until the latter starts to dominate the other two factors, namely, $(m-v+1)$ and $\frac{1}{2}(\sqrt{1+2m\varphi}-1)$. The theorem also brings out the tension between the persistence parameter v and the quality of approximation. At smaller values of v, the algorithm mines 'deeper' within each batch and so we expect fewer errors with respect to the true top-k epispodes. On the other hand, deeper mining within batches is computationally more intensive, with the required effort approaching that of exact top-k mining as v approaches 1.

Finally, we use *Theorem 3* to derive error-bounds for three special cases; first for $v=1$, when the batchwise threshold is same as that for exact top-k mining as per *Theorem 1*; second for $v=m$, when the batchwise threshold is simply the batch frequency of the k^{th} most-frequent episode in the batch; and third, for $v = \left\lfloor \frac{m+1}{2} \right\rfloor$, when the batchwise threshold lies midway between the thresholds of the first two cases. (Proofs are detailed in [10]).

Corollary 1: Let every batch $B_{s'} \in W_s$ have a top-k separation of (φ, ϵ) and let W_s contain at least $m \geq 2$ batches. Let $\mathcal{P}$ denote the set of all (v,k)-persistent episodes over W_s. If we have $|\mathcal{P}| \geq k$, then the maximum error-rate in the top-k episodes derived from $\mathcal{P}$, for three different choices of v, is given by:

1) $\left(\frac{\epsilon k m}{m-1}\right)$ for $v=1$, if $\frac{\varphi}{2} > (m-1)$
2) $(\epsilon k m)$ for $v=m$, if $\frac{\varphi}{2} > 1$

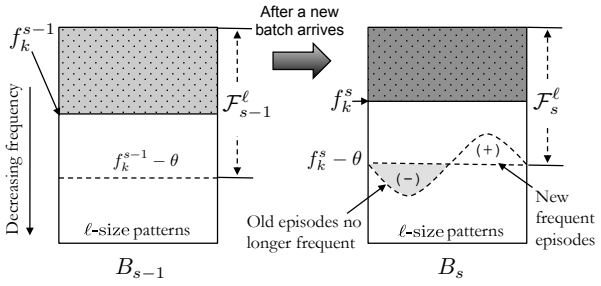


Fig. 3. The set of frequent episodes can be incrementally updated as new batches arrive.

3) $\left(\frac{4\epsilon k m^2}{m^2-1}\right)$ for $v = \lfloor \frac{m+1}{2} \rfloor$, if $\frac{\varphi}{2} > \frac{1}{m}\left\lceil \frac{m-1}{2} \right\rceil \left\lceil \frac{m+1}{2} \right\rceil$

Using $v = 1$ we make roughly ϵk errors by considering only persistent episodes for the final output, while the same batchwise threshold can give us the exact top-k as per *Theorem 1*. On the other hand, at $v = m$, the batchwise thresholds are higher (the algorithm will run faster) but the number of errors grows linearly with m. Note that the (φ, ϵ)-separation needed for $v = 1$ is much higher than for $v = m$. The case of $v = \left\lceil \frac{m+1}{2} \right\rceil$ lies in-between, with roughly $4\epsilon k$ errors under reasonable separation conditions.

V. Algorithm

In this section we present an efficient algorithm for incrementally mining episodes with frequency at least $(f_k^s - \theta)$ in batch B_s, for each batch in the stream. The threshold parameter θ is set to $2(m-v)\Delta$ for mining (v,k)-persistent episodes (see *Theorem 2*) and to $2(m-1)\Delta$ for exact top-k mining (see *Theorem 1*).

A trivial (brute-force) algorithm to find all episodes with frequency greater than $(f_k^s - \theta)$ in B_s is as follows: Apply an Apriori-based episode mining algorithm (e.g. [3], [8]) on batch B_s. If the number of episodes of size-ℓ is less than k, the support threshold is decreased and the mining repeated until at least k ℓ-size episodes are found. At this point f_k^s is known. The mining process is then repeated once more with the frequency threshold $(f_k^s - \theta)$. Doing this entire procedure for every new batch is expensive and wasteful. After seeing the first batch of the data, whenever a new batch arrives we have information about the episodes that were frequent in the previous batch. This can be exploited to incrementally and efficiently update the set of frequent episodes in the new batch. The intuition is that the frequencies of most episodes do not change much from one batch to the next. As a result only a small number of previously frequent episodes fall below the new support threshold in the new batch; similarly, some new episodes may become frequent. This is illustrated in Figure 3. In order to efficiently find these sets of episodes, we need to maintain additional information that allows us to avoid full-blown candidate generation in each batch. We show that this state information is a by-product of an Apriori-based algorithm and therefore any extra processing is unnecessary.

Frequent episodes are discovered level-wise, in ascending order of episode-size. An Apriori procedure alternates between counting and candidate generation. First a set $\mathcal{C}^i$ of candidate i-size episodes is determined by combining frequent $(i-1)$-size episodes from the previous level. Then the data is scanned for determining frequencies of the candidates, from which the frequent i-size episodes are obtained. We note that all candidate episodes that are not frequent constitute the negative border of the frequent lattice [11]. This is because, a candidate is generated only when all its subepisodes are frequent. The usual approach is to discard the border. For our purposes, the border contains information required to identify changes in the frequent episodes from one batch to the next[2].

The pseudocode for incrementally mining frequent episodes in batches is listed in *Algorithm 1*. The inputs to the algorithm are: (i) Number k of top episodes desired, (ii) New incoming batch B_s of events in the stream, (iii) Lattice of frequent $(\mathcal{F}_{s-1}^*)$ and border episodes $(\mathcal{B}_{s-1}^*)$ from previous batch, and (iv) threshold parameter θ. Frequent and border episodes of size-i, with respect to frequency threshold $f_k^s - \theta$, are denoted by the sets $\mathcal{F}_s^i$ and $\mathcal{B}_s^i$ respectively ($\mathcal{F}_s^*$ and $\mathcal{B}_s^*$ denote the corresponding sets for all episode sizes).

For the first batch B_1 (lines 1–3) the top-k episodes are found by a brute-force method, i.e., by mining with a progressively lower frequency threshold until at least k episodes of size ℓ are found. Once f_k^1 is determined, $\mathcal{F}_1^*$ and $\mathcal{B}_1^*$ are obtained using an Apriori procedure.

For subsequent batches, we do not need a brute-force method to determine f_k^s. Based on *Remark 1*, if $\theta \geq 2\Delta$, $\mathcal{F}_{s-1}^{\ell}$ from batch B_{s-1} contains every potential top-k episode of the next batch B_s. Therefore simply updating the counts of all episodes in $\mathcal{F}_{s-1}^{\ell}$ in the new batch B_s and picking the k^{th} highest frequency gives f_k^s (lines 4–6). To update the lattice of frequent and border episodes (lines 7–24) the procedure starts from the bottom (size-1 episodes). The data is scanned to determine the frequency of new candidates together with the frequent and border episodes from the lattice (line 11). In the first level (episodes of size 1), the candidate set is empty. After counting, the episodes from $\mathcal{F}_{s-1}^{\ell}$ that continue to be frequent in the new batch are added to $\mathcal{F}_s^{\ell}$ (lines 13–14). But if an episode is no longer frequent it is marked as a border set and all its super episodes are deleted (lines 15–17). This ensures that only border episodes are retained in the lattice. In the border and new candidate sets, any episode that is found to be frequent is added to both $\mathcal{F}_s^i$ and $\mathcal{F}_{new}^i$ (lines 18–21). The remaining infrequent episodes belong to the border because, otherwise, they would have at least one infrequent subepisode and would have been deleted at a previous level; hence, these infrequent episodes are added to $\mathcal{B}_s^{\ell}$ (lines 22–23).

Finally, the candidate generation step (line 24) is required to fill out the missing parts of the frequent lattice. We want to avoid a full blown candidate generation. Note that if an episode is frequent in B_{s-1} and B_s then all its subepisodes are also frequent in both B_s and B_{s-1}. Any new episode $(\notin \mathcal{F}_{s-1}^{\ell} \cup \mathcal{B}_{s-1}^{\ell})$ that turns frequent in B_s, therefore, must have at least one subepisode that was not frequent in B_{s-1}

[2]Border sets were employed by [11] for efficient mining of dynamic databases. Multiple passes over older data are needed for any new frequent itemsets, which is not feasible in a streaming context.

Algorithm 1 Persistent episode miner: Mine top-k v-persistent episodes.

Input: Number k of top episodes; New batch B_s of events; Current lattice of frequent & border episodes $(\mathcal{F}^*_{s-1}, \mathcal{B}^*_{s-1})$; Threshold parameter θ (set to $2(m-v)\Delta$ for (v,k)-persistence, $2(m-1)\Delta$ for exact top-k)
Output: Lattice of frequent and border episodes $(\mathcal{F}^*_s, \mathcal{B}^*_s)$ after B_s

1: **if** $s = 1$ **then**
2: Determine f^1_k (brute force)
3: Compute $(\mathcal{F}^*_1, \mathcal{B}^*_1) \leftarrow \mathbf{Apriori}(B_1, \ell, f^1_k - \theta)$
4: **else**
5: **CountFrequency**$(\mathcal{F}^\ell_{s-1}, B_s)$
6: Determine f^s_k (based on episodes in $\mathcal{F}^\ell_{s-1}$)
7: Initialize $\mathcal{C}^1 \leftarrow \phi$ (new candidates of size 1)
8: **for** $i \leftarrow 1, \ldots, \ell$ **do**
9: Initialize $\mathcal{F}^i_s \leftarrow \phi$, $\mathcal{B}^i_s \leftarrow \phi$
10: Initialize $\mathcal{F}^i_{new} \leftarrow \phi$ (new frequent episodes of size i)
11: **CountFrequency**$(\mathcal{F}^i_{s-1} \cup \mathcal{B}^i_{s-1} \cup \mathcal{C}^i, B_s)$
12: **for** $\alpha \in \mathcal{F}^i_{s-1}$ **do**
13: **if** $f^s(\alpha) \geq f^s_k - \theta$ **then**
14: Update $\mathcal{F}^i_s \leftarrow \mathcal{F}^i_s \cup \{\alpha\}$
15: **else**
16: Update $\mathcal{B}^i_s \leftarrow \mathcal{B}^i_s \cup \{\alpha\}$
17: Delete super-episodes of α from $(\mathcal{F}^*_{s-1}, \mathcal{B}^*_{s-1})$
18: **for** $\alpha \in \mathcal{B}^i_{s-1} \cup \mathcal{C}^i$ **do**
19: **if** $f^s(\alpha) \geq f^s_k - \theta$ **then**
20: Update $\mathcal{F}^i_s \leftarrow \mathcal{F}^i_s \cup \{\alpha\}$
21: Update $\mathcal{F}^i_{new} \leftarrow \mathcal{F}^i_{new} \cup \{\alpha\}$
22: **else**
23: Update $\mathcal{B}^i_s \leftarrow \mathcal{B}^i_s \cup \{\alpha\}$
24: $\mathcal{C}^{i+1} \leftarrow$ **GenerateCandidate**$(\mathcal{F}^i_{new}, \mathcal{F}^i_s)$
25: **return** $(\mathcal{F}^*_s, \mathcal{B}^*_s)$

but is frequent in B_s. All such episodes are listed in $\mathcal{F}^i_{new}$. Hence the candidate generation step (line 24) for the next level generates only candidates with atleast one subepisode $\in \mathcal{F}^i_{new}$. This reduces the number of candidates generated at each level without compromising completeness of the results. The space and time complexity of candidate generation is now $O(|\mathcal{F}^i_{new}|.|\mathcal{F}^i_s|)$ instead of $O(|\mathcal{F}^i_s|^2)$ and in most practical cases $|F^i_{new}| \ll |\mathcal{F}^i_s|$. Later in the experiments section, we show how border-sets help our algorithm run very fast.

For a window W_s ending in the batch B_s, the set of output episodes can be obtained by picking the top-k most frequent episodes from the set $\mathcal{F}^\ell_s$. Each episode also maintains a list that stores its batch-wise counts is last m batches. The window frequency is obtained by adding these entries together. The output episodes are listed in decreasing order of their window counts.

Example 2: In this example we illustrate the procedure for incrementally updating the frequent episodes lattice as a new batch B_s is processed (see Figure 4).

Figure 4(A) shows the lattice of frequent and border episodes found in the batch B_{s-1} (The figure does not show all the subepisodes at each level, only some of them). $ABCD$ is a 4-size frequent episode in the lattice. In the new batch B_s, the episode $ABCD$ is no longer frequent. The episode $CDXY$ appears as a new frequent episode. The episode lattice in B_s is shown in Figure 4(B).

In the new batch B_s, AB falls out of the frequent set. AB

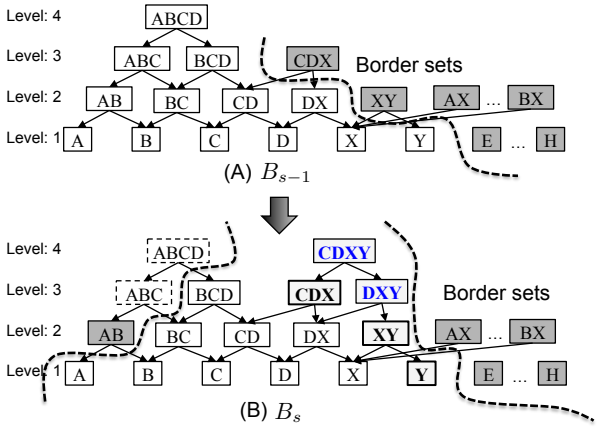


Fig. 4. Incremental lattice update for the next batch B_s given the lattice of frequent and border episodes in B_{s-1}.

now becomes the new border and all its super-episodes namely ABC, BCD and $ABCD$ are deleted from the lattice.

At level 2, the border episode XY turns frequent in B_s. This allows us to generate DXY as a new 3-size candidate. At level 3, DXY is also found to be frequent and is combined with CDX which is also frequent in B_s to generate $CDXY$ as a 4-size candidate. Finally at level 4, $CDXY$ is found to be frequent. This shows that border sets can be used to fill out the parts of the episode lattice that become frequent in the new data.

VI. Results

A. Experimental Setup

We show experimental results on one synthetic and two real data streams, from two very different domains: experimental neuroscience and telecom networks. In neuroscience, we consider (1) synthetic neuronal spike-trains based on mathematical models of interconnected neurons, with each neuron modeled as an inhomogeneous Poisson process [8], and (2) real neuronal spiking activity from dissociated cortical cultures, recorded using multi-electrode arrays at Steve Potter's lab, Georgia Tech [12]. The third kind of data we consider are call data records from a large European telecom network. Each record describes whether the associated call was voice or data, an international or local call, in-network or out-of-network, and of long or short duration. Pattern discovery over such data can uncover hidden, previously unknown, trends in usage patterns, evolving risk of customer churn, etc. . The data length in terms of number of events and the alphabet size of each of these datasets is shown in Table II(a). Table II(b) gives the list of parameters and the values used in experiments that we do not explicitly mention in the text.

As mentioned earlier, we are not aware of any streaming algorithms that directly address top-k *episode* mining over sliding windows of data. For our experiments, we compare persistent episode miner against two methods from itemsets mining literature [5] (after adapting them for episodes): one that fixes batchwise thresholds based on Chernoff-bounds under an *iid* assumption over the event stream, and the second based on a sliding window version of the Lossy Counting

TABLE II
EXPERIMENTAL SETUP.

(a) Datasets used in the experiments.

Dataset	Alphabet-size	Data-length
Call-logs	8	42,030,149
Neuroscience	58	12,070,634
Synthetic	515	25,295,474

(b) Parameter set-up.

Parameter	Value
k (in Top-k episodes)	25
Number of batches in a window, m	10
Batch-size (as number of events)	10^6
Error parameter (applicable to Chernoff-based and Lossy counting methods)	0.001
Parameter v (applicable to Persistence miner)	$m/2$

algorithm [4]. We modify the support threshold computation using chernoff bound for episodes since the total number of distinct itemsets is bounded by the size of the largest transaction while for episodes it is the alphabet size that determines this.

Estimating Δ dynamically: Δ is a critical parameter for our persistent episode miner. But unfortunately the choice of the correct value is highly data-dependent and the characteristics of the data can change over time. One predetermined value of Δ cannot be provided in any intuitive way. Therefore we estimate Δ from the frequencies of ℓ-size episodes in consecutive batches by computing the change in frequency of episodes and using the 75^{th} percentile as the estimate. We avoid using the maximum change as it tends to be noisy.

B. Quality of top-k mining

We present aggregate comparisons of the three competing methods in Table III. These datasets provide different levels of difficulty for the mining algorithms. Tables III(a) & III(c) shows high f-scores[3] for the synthetic and real neuroscience data for all methods (Our method performs best in both cases). On the other hand we find that all methods report very low f-scores on the call-logs data (see Table III(b)). The characteristics of this data does not allow one to infer window top-k from batches (using limited computation). But our proposed method nearly doubles the f-score with identical memory and CPU usage on this real dataset. It may be noteworthy to mention that the competing methods reported close to 100% accuracies but they do not perform that well on more realistic datasets. In case of the synthetic data (see Table III(c) the characteristics are very similar to that of the neuroscience dataset.

C. Computation efficiency comparisons

Table III shows that we do better than both competing methods in most cases (and never significantly worse than either) with respect to time and memory.

[3] $\mathrm{f_{score}} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$

TABLE III
AGGREGATE PERFORMANCE COMPARISON OF DIFFERENT ALGORITHMS.

(a) Dataset: Neuroscience, Size of episodes = 4

Top-k Miner	F Score	Runtime (sec)	Memory (MB)
Chernoff-bound based	92.0	456.0	251.39
Lossy-counting based	92.0	208.25	158.5
Persistent episode	97.23	217.64	64.51

(b) Dataset: Call-logs, Size of episodes = 6

Top-k Miner	F Score	Runtime (sec)	Memory (MB)
Chernoff-bound based	32.17	11.87	66.14
Lossy-counting based	24.18	3.29	56.87
Persistent episode	49.47	3.34	67.7

(c) Dataset: Synthetic data, Size of episodes = 4

Top-k Miner	F Score	Runtime (sec)	Memory (MB)
Chernoff-bound based	92.7	14.91	43.1
Lossy-counting based	92.7	6.96	32.0
Persistent episode	96.2	4.98	34.43

1) Effect of parameters on performance: In Figure 5, we see all three methods outperform the reference method (the standard multi-pass apriori based miner) by at least an order of magnitude in terms of both run-times and memory.

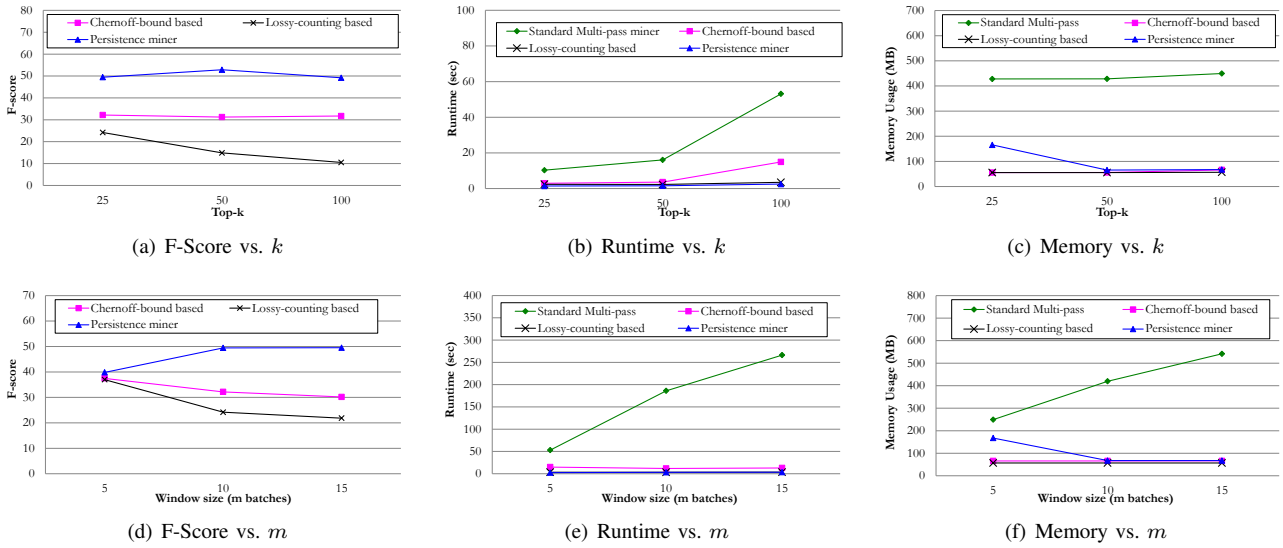
In Figure 5(a)-(c), we show the effect of increasing k on all the methods. The accuracy of Lossy-counting algorithm drops with increase in k, while that of Chernoff based method and Persistence miner remain unchanged. Persistence miner has lower runtimes for all choices of k while having comparable memory footprint as the other two methods.

With increasing window-size ($m = 5, 10, 15$ and $batch - size = 10^6$ events), we observe better f-scores for Persistence miner but this increase is not significant enough and can be caused by data characterisitcs alone. The runtimes and memory of Persistence miner remain nearly constant. This is important for streaming algorithms as the runtimes and memory of the standard multi-pass algorithm increases (roughly) linearly with window size.

2) Utility of Border Sets: For episodes with slowly changing frequency we show in Section V that using border-sets to incrementally update the frequent episodes lattice results in an order complexity of $O(|\mathcal{F}^i_{new}|.|\mathcal{F}^i_s|)$ instead of $O(|\mathcal{F}^i_s|^2)$ for candidate generation and in most practical cases $|F^i_{new}| \ll |\mathcal{F}^i_s|$. Table IV demonstrates the speed-up in runtime achieved by using border-set. We implemented two versions of our Persistence miner. In one we utilize border-sets to incrementally update the lattice where as in other we rebuild the frequent lattice from scratch in every new batch. The same batch-wise frequency thresholds used as dictated by Theorem 2. We run the experiment on our call-logs dataset and for various parameter settings of our algorithm we observe a speed-up of $\approx 4\times$ resulting from use of border-sets.

D. Adapting to Dynamic data

In this experiment we show that Persistence miner adapts faster than the competing algorithms to changes in underlying data characteristics. We demonstrate this using synthetic data generated using the multi-neuronal simulator based [8].



(a) F-Score vs. k (b) Runtime vs. k (c) Memory vs. k

(d) F-Score vs. m (e) Runtime vs. m (f) Memory vs. m

Fig. 5. Performance with different parameters (Dataset: Call logs)

TABLE IV
UTILITY OF BORDER SET.

(a) Dataset: Call-logs, Size of episodes = 6, Parameter $v = m/2$

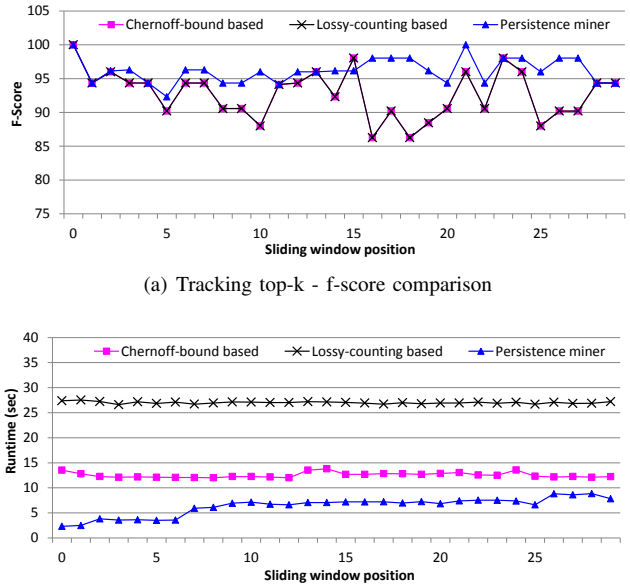
Window size	Runtime (border-set)	Runtime (no border-set)	Memory (border-set)	Memory (no border-set)
5	2.48	12.95	67.55	66.21
10	3.13	11.41	67.7	66.67
15	3.47	13.76	67.82	67.02

(b) Dataset: Call-logs, Size of episodes = 6, window size $m = 10$

Parameter v	Runtime (border-set)	Runtime (no border-set)	Memory (border-set)	Memory (no border-set)
0	2.78	11.98	67.7	66.67
5	3.13	11.41	67.7	66.67
10	3.21	10.85	67.69	57.5

The simulation model was adapted to update the connection strengths dymanically while generating synthetic data. This allowed us to change the top-k episodes slowly over the length of simulated data. We embedded 25 randomly picked patterns with time-varying arrival rates. Figure 6(a) shows the performance of the different methods in terms of f-score computed after arrival of each new batch of events but for top-k episodes in the window. The ground truth is again the output of the standard multi-pass apriori algorithm that is allowed access to the entire window of events. The f-score curves of the both the competing methods almost always below that of the Persistence miner. While the runtimes for Persistence miner are always lower than those of the competing methods (see Figure 6(b). Lossy counting based methods is the slowest at error parameter set to 0.0001.

The main reason of better tracking in the case of Persistence miner is that the output of the algorithm filters out all non-(v, k) persistent episodes. This acts in favor of Persistence miner as the patterns most likely to gather sufficient support to be in the top-k are also likely to be persistent. The use of border-sets in Persistence miner explains the lower runtimes.



(a) Tracking top-k - f-score comparison

(b) Tracking top-k - runtime

Fig. 6. Comparison of different methods in tracking top-k episodes over dynamically changing event stream (Parameters used: k = 25, m = 10, Persistence miner: v=0.5, alg 6,7: error = 1.0e-4)

E. Correlation of f-score with theoretical error

Can we compute theoretical errors (data-dependent quantity) and guess how well we perform? This could be invaluable in a real-world streaming data setting.

In this experiment we try to establish the usefulness of the theoretical analysis proposed in the paper. The main power of the theory is to predict the error in the output set at the end of each batch. Unlike other methods we compute the error bounds

using the data characteristics and is dynamically updated as new data arrives. The error guarantees of both Lossy counting and Chernoff based methods are static.

In Figure 7, we plot the error bound using Theorem 3 and the f-score computed with respect to the reference method (standard multi-pass apriori) in a 2D histogram. According to the theory different pairs of (ϕ, ϵ) output a different error bound in every batch. In our experiment we pick the smallest error bound in the alloable range of ϕ and corresponding ϵ in each batch and plot it with the corresponding f-score. The histogram is expected to show negative correlation between f-score and our error predicted error bound i.e. the predicted error for high-f-score top-k results should be low and vise versa. The correlation is not very evident in the plot. The histogram shows higher density in the left top part of the plot, which is a mild indication that high f-score has a corresponding low error predicition.

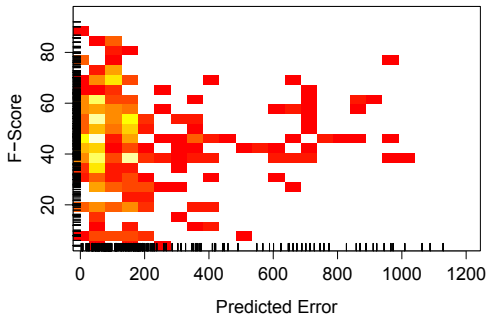


Fig. 7. 2D Histogram of predicted error vs. f-score. (Dataset: Call logs)

VII. Related work

The literature for streaming algorithms for pattern discovery is dominated by techniques from the frequent itemset mining context [4]–[7], [13]–[17], and we are unaware of any algorithms for episode mining over event streams.

Manku and Motwani [4] proposed the lossy counting Algorithm for approximate frequency counting in the landmark model (i.e., all events seen until the current-time constitute the window of interest). One of its main drawbacks is that the algorithm must essentially operate at a threshold of ϵ to provide ϵ error guarantees, which is impractical in many real-world scenarios. Karp et al. [13] also propose a one pass streaming algorithm for finding frequent items, and these ideas were extended to itemsets by Jin and Agrawal [16], but all these methods require even more space than lossy counting. Mendes et al. [18] extend the pattern growth algorithm (PrefixSpan) [19] for mining sequential patterns to incorporate the idea of lossy counting. Chang and Lee [15] and Wong and Fu [5] extend lossy counting to sliding windows and top-k setting, respectively. New frequency measures for itemsets over streams have also been proposed (e.g., Calders et al. [6] Lam et al. [17]) but these methods are heavily specialized toward the itemset context and it is not obvious how to extend them to accommodate episodes in a manner that supports both candidate generation and counting.

VIII. Conclusions

While episode mining in offline multi-pass scenarios has been researched [3], [8], this paper is the first to explore ways of doing both counting and candidate generation efficiently in a streaming setting. We have presented algorithms that can operate at as high frequency thresholds as possible and yet give certain guarantees about frequent output patterns. One of our directions of future work is to further the idea of automatic 'streamification' of algorithms whereby black-box mining algorithms that operate in levelwise fashion can exploit the approximation guarantees and border set efficiencies introduced here. We also aim to increase the expressiveness of our episode mining formulation; for instance, by combining episode mining with itemsets we will be able to mine more generalized partial order episodes in a stream.

References

[1] S. Muthukrishnan, "Data streams: Algorithms and applications," *Foundations and Trends in Theoretical Computer Science*, vol. 1, no. 2, 2005.

[2] D. Patnaik *et al.*, "Discovering excitatory networks from discrete event streams with applications to neuronal spike train analysis," in *Proc. of The 9th IEEE Intl. Conf. on Data Mining (ICDM)*, 2009.

[3] H. Mannila *et al.*, "Discovery of frequent episodes in event sequences," *Data Mining and Knowl. Dsc.*, vol. 1, no. 3, pp. 259–289, 1997.

[4] G. S. Manku and R. Motwani, "Approximate frequency counts over data streams," in *Proc. of the 28th Intl. Conf on Very Large Data Bases (VLDB)*, 2002, pp. 346–357.

[5] R. C.-W. Wong and A. W.-C. Fu, "Mining top-k frequent itemsets from data streams," *Data Mining and Knowl. Dsc.*, vol. 13, pp. 193–217, 2006.

[6] T. Calders *et al.*, "Mining frequent itemsets in a stream," in *Proc. of the 7th IEEE Intl. Conf. on Data Mining (ICDM)*, 2007, pp. 83–92.

[7] J. Cheng *et al.*, "A survey on algorithms for mining frequent itemsets over data streams," *Knowl. and Info. Systems*, vol. 16, no. 1, pp. 1–27, 2008.

[8] A. Achar *et al.*, "Discovering injective episodes with general partial orders," *Data Mining and Knowl. Dsc.*, vol. 25, no. 1, pp. 67–108, 2012.

[9] J. Wang *et al.*, "TFP: An efficient algorithm for mining top-k frequent closed itemsets," *IEEE Trans. on Knowledge and Data Engg.*, vol. 17, no. 5, pp. 652–664, 2005.

[10] D. Patnaik *et al.*, "Streaming algorithms for pattern discovery over dynamically changing event sequences," *CoRR*, vol. abs/1205.4477, 2012.

[11] Y. Aumann *et al.*, "Borders: An efficient algorithm for association generation in dynamic databases," *J. of Intl. Info. Syst. (JIIS)*, no. 1, pp. 61–73, 1999.

[12] D. A. Wagenaar *et al.*, "An extremely rich repertoire of bursting patterns during the development of cortical cultures," *BMC Neuroscience*, 2006.

[13] R. M. Karp *et al.*, "A simple algorithm for finding frequent elements in streams and bags," *ACM Trans. Database Syst.*, vol. 28, pp. 51–55, 2003.

[14] J. H. Chang and W. S. Lee, "Finding recent frequent itemsets adaptively over online data streams," in *Proc. of the 9th ACM SIGKDD Intl. Conf. on Knowledge discovery and data mining (KDD)*, 2003, pp. 487–492.

[15] ——, "A sliding window method for finding recently frequent itemsets over online data streams," *J. Inf. Sci. Eng.*, vol. 20, no. 4, pp. 753–762, 2004.

[16] R. Jin and G. Agrawal, "An algorithm for in-core frequent itemset mining on streaming data," in *Proc. of the 5th IEEE Intl. Conf. on Data Mining (ICDM)*, 2005, pp. 210–217.

[17] H. T. Lam *et al.*, "Online discovery of top-k similar motifs in time series data," in *SIAM Intl. Conf. of Data mining*, 2011, pp. 1004–1015.

[18] L. Mendes, B. Ding, and J. Han, "Stream sequential pattern mining with precise error bounds," in *Proc. of the 8th IEEE Intl. Conf. on Data Mining (ICDM)*, 2008, pp. 941–946.

[19] J. Pei *et al.*, "Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth," in *Proc. of the 17th Intl. Conf. on Data Engg. (ICDE)*, 2001, pp. 215–.