Under consideration for publication in Knowledge and Information Systems

A General Streaming Algorithm for Pattern Discovery

Debprakash Patnaik[1], Srivatsan Laxman[2], Badrish Chandramouli[3], and Naren Ramakrishnan[4]
[1]Amazon.com, Seattle, WA 98109, USA;
[2]Microsoft Research, Bangalore 560080, India;
[3]Microsoft Research, Redmond, WA, USA;
[4]Department of Computer Science, Virginia Tech, VA 24061, USA

Abstract. Discovering frequent patterns 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 patterns over a window of recent events in the stream. We derive approximation guarantees for our algorithm in terms of: (i) the separation of frequent patterns 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.

Keywords: Event Sequences; Data Streams; Frequent patterns; Pattern Discovery; Streaming Algorithms; Approximation Algorithms

1. Introduction

Application contexts (Patnaik, Marwah, Sharma and Ramakrishnan, 2009; Ramakrishnan, Patnaik and Sreedharan, 2009) in telecommunications, neuroscience, sustainability, and intelligence analysis feature massive data streams (Muthukrishnan, 2005) with 'firehose'-like rates of arrival. In many cases, we need to analyze such streams at speeds comparable to their generation rate. In neuroscience,

Received Feb 01, 2013
Revised Feb 01, 2013
Accepted Feb 01, 2013

one goal is to track spike trains from multi-electrode arrays (Patnaik, Laxman and Ramakrishnan, 2009) 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 patterns (i.e., a succession of events occurring frequently, but not necessarily consecutively (Mannila et al., 1997)) from dynamic and evolving streams.

Algorithms for pattern mining over streams have become increasingly popular over the recent past (Manku and Motwani, 2002; Wong and Fu, 2006; Calders et al., 2007; Cheng et al., 2008). Manku and Motwani (Manku and Motwani, 2002) 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 (Wong and Fu, 2006) 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. 6 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 patterns.

Our specific contributions are as follows:

- We present the *first* general algorithm for mining patterns 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.
- Although our work is geared towards pattern mining, we adopt a black-box model of a pattern 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 illustrate here this generality of our approach by focusing on two pattern classes—itemsets and episodes—prevalent in data mining research.
- Devoid of any statistical assumptions on the stream (e.g., independence of event symbols or otherwise), we develop a novel error characterization for streaming patterns 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.
- 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 pattern-mining adaptations of two typical algorithms (Wong and Fu, 2006) from the streaming itemsets literature.

2. Preliminaries

We consider the data mining task of discovering frequent patterns over a time-stamped sequence of data records (Laxman and Sastry, 2006). The *data se-*

quence is denoted as $\langle (e_1, \tau_1), (e_2, \tau_2), \ldots, (e_n, \tau_n) \rangle$, where each data record e_i, may represent different kinds of data depending on the class of patterns being considered. For example, e_i denotes a *transaction* in the frequent itemsets setting (Agrawal and Srikant, 1994), an *event* in the frequent episodes setting (Mannila et al., 1997), a *graph* in the frequent subgraph mining setting (Yan and Han, 2003), etc. The techniques developed in this paper are relevant in all of these settings. In our experimental work, we focus on two concrete classes of patterns; frequent episodes and frequent itemsets; we briefly summarize the associated formalisms for these in the paragraphs below.

Frequent Episodes:

In the framework of frequent episodes (Mannila et al., 1997), 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 episodes* 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* (Achar et al., 2012) 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.

Frequent Itemsets:

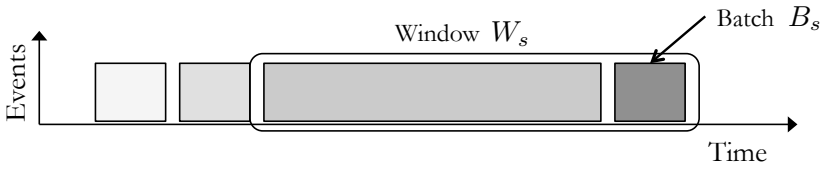
The frequent itemsets mining framework is concerned with the classical market basket problem (Agrawal and Srikant, 1994), where each data record e_i can be viewed as a transaction of *items* purchased at a grocery store. Frequent itemsets will then refer to groups of itemsets frequently purchased together in the given data set of transactions.

In general, the task in frequent pattern discovery is to find all patterns whose frequency exceeds a user-defined threshold. Apriori-style level-wise[1] algorithms (Agrawal and Srikant, 1994; Mannila et al., 1997; Yan and Han, 2003; Achar et al., 2012) are typically applicable in this setting. An important variant is top-k pattern mining (see (Wang et al., 2005) for definitions in the itemsets mining context), where, rather than a frequency threshold, the user supplies the *number* of most frequent patterns needed.

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

The number of top-k ℓ-size patterns can exceed k, although the number of ℓ-size patterns with frequencies strictly greater than f^k is at most $(k-1)$. In

[1] Level-wise algorithms start with patterns of size 1 and with each increasing level estimate frequent patterns of the next size.



[t]

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

general, top-k mining can be difficult to solve without knowledge of a good lower-bound for f^k; for relatively short data 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.

3. Problem Statement

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

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

Our goal is to find all patterns that were frequent in the recent past; for this, we consider a sliding window model[2] for the *window of interest*. In this model, the user wants to determine patterns that were frequent over a (historical) window of fixed-size terminating at the current time-tick. As new records arrive in the stream, the user's window of interest shifts, and the data mining task is to next report the frequent patterns 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 pattern discovery over the window of interest. We organize the records 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 records 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 $(s = 1, 2, \ldots)$.

The frequency of a pattern 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.

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

Definition 2 (Window Frequency). The frequency of a pattern 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 $f^{W_s}(\alpha) = \sum_{B_j \in W_s} f^j(\alpha)$.

[2] Other models such as the landmark and time-fading models have also been studied (Cheng et al., 2008) but we do not consider them here.

[t]

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.*

[t]

Table 1. Window frequencies in *Example 1.*

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

In summary, we are given a data 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 patterns (ℓ), the desired number of most frequent patterns (k) and the problem is to discover the top-k patterns 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 records in the stream, find all ℓ-size patterns 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-size pattern 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 patterns in each batch with corresponding batch frequencies are listed in Fig. 2. The corresponding window frequencies (sum of each patterns' batch frequencies) are listed in Table 1. The top-2 patterns in B_1 are (PQRS) and (WXYZ). Similarly (EFGH) and (IJKL) are the top-2 patterns 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 patterns would 'fly below the radar' and go undetected if we considered only the top-2 patterns in every batch as candidates for the top-2 patterns over W. This example can be easily generalized to any number of batches and any k.

It is also clear that the size of the batches can play a critical role with regards to the quality of approximation of top-k. At one end of the spectrum the batch-size can be as large as the window. Ee would get exact top-k results but such a scheme would be obviously impractical. At the other end the algorithm could consume events one at a time (batch-size = 1) updating the top-k results over the window in an online fashion. However, we expect the quality of approximation to be poor in this case since no local pattern statistics can be estimated reliably over batches of size=1. Thus, we suggest using batches of sufficient size, ofcourse, limited by the number of events that can be stored and processed in-memory. This will allow us to exploit the slowly changing statistics across batches to get better top-k approximations over the window.

Example 1 highlights the main challenge in the streaming top-k mining problem: we can only store/process the most recent batch of records 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 patterns (both frequent and infrequent) in every batch since the pattern space is typically very large. This brings us to the question of which patterns to track in every batch – how deep must we search within each batch for patterns that have potential to become top-k over the window? We develop the formalism needed to answer this question.

4. Persistence and Top-k Approximation

We identify two important properties of the underlying data 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 pattern, 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 records arriving per batch, it is often much smaller in-practice.

Definition 4 (Top-k Separation of (φ, ϵ)). A batch B_s of records is said to have a *top-k separation of* (φ, ϵ), $\varphi \geq 0$, $\epsilon \geq 0$, if it contains at most $(1+\epsilon)k$ patterns 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 pattern 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 patterns are relative to the rest of the patterns. We expect to see roughly k patterns 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 patterns, 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 patterns (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 pattern 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 patterns 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

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

Proof. There exist at least k patterns in B_s with batch frequency greater than or equal to f_k^s (by definition). Hence, there exist at least k patterns in B_{s+1} with batch frequency greater than or equal to $(f_k^s - \Delta)$ (since frequency of any pattern can decrease by at most Δ going from B_s to B_{s+1}). Hence we must have $f_k^{s+1} \geq (f_k^s - \Delta)$. Similarly, there can be at most $(k-1)$ patterns in B_{s+1} with batch frequency strictly greater than $(f_k^s + \Delta)$. Hence we must also have $f_k^{s+1} \leq (f_k^s + \Delta)$. □

The above lemma follows directly from: (i) there are at least k patterns with frequencies no less than f_k^s, and (ii) the batch frequency of any pattern 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 a pattern 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 pattern 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 $f^{s+r}(\alpha) < f_k^{s+r} + 2|r|\Delta$

Proof. Since Δ is the maximum rate of change, we have $f^{s+r}(\alpha) \geq (f^s(\alpha) - |r|\Delta)$ and from Lemma 1, we have $f_k^{s+r} \leq (f_k^s + |r|\Delta)$. Therefore, if $f^s(\alpha) \geq f_k^s$, then

$$f^{s+r}(\alpha) + |r|\Delta \geq f^s(\alpha) \geq f_k^s \geq f_k^{s+r} - |r|\Delta$$

which implies $f^{s+r}(\alpha) \geq f_k^{s+r} - 2|r|\Delta$. Similarly, if $f^s(\alpha) < f_k^s$, then

$$f^{s+r}(\alpha) - |r|\Delta \leq f^s(\alpha) < f_k^s \leq f_k^{s+r} + |r|\Delta$$

which implies $f^{s+r}(\alpha) < f_k^{s+r} + 2|r|\Delta$. □

Lemma 2 gives us a way to track patterns 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 patterns 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 patterns of B_{s+1} must have batch frequencies of at least $(f_k^s - 2\Delta)$ in B_s.

Proof. Assume that we know $f_k^{s'}$ for the batch $B_{s'}$. If a pattern α is to belong to the set of top-k frequent patterns in the batch $B_{s'+r}$, then $f^{s'+r}(\alpha) \geq f_k^{s'+r}$ (where $f_{s'+r}(\alpha)$ and $f_k^{s'+r}$ are unknown for $r > 0$).

Substituting $s = s' + r$ in Lemma 2, we get $f^{s'}(\alpha) \geq f_k^{s'} - 2|r|\Delta$. □

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

Theorem 1 (Exact Top-k over W_s). A pattern, α, can be a top-k pattern 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. To prove this, we show that if a pattern fails this threshold in even one batch of W_s, then there exist k or more patterns which can have a greater window frequency. Consider a pattern 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 pattern of $B_{s'}$. Consider two patterns α and β such that in a batch $B_{s'}$ $f^{s'}(\alpha) \geq f_k^{s'}$ and $f^{s'}(\beta) < f_k^{s'} - 2(m-1)\Delta$. 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 pattern α of $B_{s'}$. Since there are at least k top-k patterns in $B_{s'}$, β cannot be a top-k pattern over the window W_s. □

Mining patterns with frequency threshold $(f_k^{s'} - 2(m-1)\Delta)$ in each batch $B_{s'}$ gives complete counts of all top-k patterns in the window W_s where m is the number of batches in the window, $f_k^{s'}$ is the frequency of the kth most frequent pattern in batch $B_{s'} \in W_s$ and Δ is the continuity parameter.

Based on *Theorem 1* we have the following simple algorithm for obtaining the top-k patterns over a window: Use a traditional level-wise approach to find all patterns 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 patterns with the k highest window frequencies over W_s. This approach is guaranteed to return the exact top-k patterns over W_s. In order to report the top-k over the next sliding window W_{s+1}, we need to consider all patterns 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 pattern mining algorithm in every batch, B_s, $s = 1, 2, \ldots$, with a frequency threshold of $(f_s^k - 2(m-1)\Delta)$.

4.1. Class of (v, k)-Persistent patterns

Theorem 1 characterizes the minimum batchwise computation needed in order to obtain the exact top-k patterns 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 patterns to track over the window to become impractically high. To address this issue, we introduce a new class of patterns called (v, k)-*persistent patterns* which can be computed efficiently by employing higher batchwise thresholds. Further, we show that these patterns can be used to approximate the true top-k patterns 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 pattern). A pattern is said to be (v,k)-*persistent* over window W_s if it is a top-k pattern in *at least* v batches of W_s.

Problem 2 (Mining (v,k)-Persistent patterns). For each new batch, B_s, of records in the stream, find all ℓ-size (v,k)-persistent patterns in the corresponding window of interest, W_s.

Theorem 2. A pattern, 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 patterns over sliding windows in the stream. The batchwise threshold required for (v,k)-persistent patterns 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 patterns 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 pattern mining algorithm to find all patterns with batch frequency of at least $(f_s^k - 2(m-v)\Delta)$. We provide more details of our algorithm later in Sec. 5. First, we investigate the quality of approximation of top-k that (v,k)-persistent patterns offer and show that the number of errors is closely related to the degree of top-k separation.

4.1.1. 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 patterns 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 patterns and an upper-bound (f_U) on the window frequencies of patterns that are *not* (v,k)-persistent (cf. *Lemmas 3 & 4*).

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

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

Proof. Consider pattern 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

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

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

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

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

Similar arguments give us the next lemma about the maximum frequency of patterns that are *not* (v, k)-persistent (Full proofs are available in (Patnaik et al., 2012)).

Lemma 4. If pattern 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:

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

Proof. Consider pattern β that is not (v, k)-persistent over W_s and let V_β denote the batches of W_s in which β is in the top-k. The window frequency of β can be written as:

$$\begin{aligned} f^{W_s}(\beta) &= \sum_{B_p \in V_\beta} f^p(\beta) + \sum_{B_q \in W_s \setminus V_\beta} f^q(\beta) \\ &< \sum_{B_p \in V_\beta} f_k^p + 2|\widehat{p}(q) - q|\Delta + \sum_{B_q \in W_s \setminus V_\beta} f_k^q \\ &= \sum_{B_{s'} \in W_s} f_k^{s'} + \sum_{B_p \in V_\beta} 2|\widehat{q}(p) - p|\Delta \end{aligned} \tag{12}$$

where $B_{\widehat{q(p)}} \in W_s \setminus V_\beta$ denotes the batch nearest B_p where β is not in the top-k. Since $|V_\beta| < v$, we must have

$$\begin{aligned} \sum_{B_p \in V_\beta} |\widehat{q}(p) - p| &\leq (1 + 2 + \cdots + (v - 1)) \\ &= \frac{1}{2} v(v + 1) \end{aligned} \tag{13}$$

Putting together (12) and (13) gives us the lemma. □

It turns out that $f_U > f_L \ \forall v,\ 1 \leq v \leq m$, and hence there is always a possibility for some patterns which are not (v,k)-persistent to end up with higher window frequencies than one or more (v,k)-persistent patterns. 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 patterns from the rest of the patterns 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 pattern in every batch, somewhere close to it is sufficient to achieve a good top-k approximation.

Definition 6 (Band Gap patterns, $\mathcal{G}_\varphi$). In any batch $B_{s'} \in W_s$, the half-open frequency interval $[f^k_{s'} - \varphi\Delta,\ f^k_{s'})$ is called the *band gap* of $B_{s'}$. The corresponding set, $\mathcal{G}_\varphi$, of *band gap patterns* over the window W_s, is defined as the collection of all patterns 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 patterns which are not (v,k)-persistent but that can still mix with (v,k)-persistent patterns are those belonging to $\mathcal{G}_\varphi$. This is stated formally in the next lemma. The proof, omitted here, can be found in (Patnaik et al., 2012).

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

Proof. If a pattern β is *not* (v,k)-persistent over W_s then there exists a batch $B_{s'} \in W_s$ where β is not in the top-k. Further, if $\beta \notin \mathcal{G}_\varphi$ then we must have $f_{s'}(\beta) < f^k_{s'} - \varphi\Delta$. Since $\varphi > 2$, β cannot be in the top-k of any neighboring batch of $B_{s'}$, and hence, it will stay below $f^k_{s'} - \varphi\Delta$ for all $B_{s'} \in W_s$, i.e.,

$$f^{W_s}(\beta) < \sum_{B_{s'} \in W_s} f^k_{s'} - m\varphi\Delta.$$

The Lemma follows from the given condition $\frac{\varphi}{2} > (1-\frac{v}{m})(m-v+1)$. $\square$

The number of patterns in $\mathcal{G}_\varphi$ is controlled by the top-k separation property, and since many of the non-persistent patterns which can mix with persistent ones must spend not one, but several batches in the band gap, the number of unique patterns 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 patterns over W_s. If $|\mathcal{P}| \geq k$, then the top-k patterns over W_s can be determined from $\mathcal{P}$ with an error of no more than $\left(\frac{\epsilon k m}{\mu}\right)$ patterns, 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 $(1+\epsilon)k$ patterns 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 patterns that can belong to the band gap of $B_{s'}$. Thus, there can be no more than a total of $\epsilon k m$ patterns 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 patterns 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 patterns over W_s with error no more than $\left(\frac{\epsilon k m}{\mu}\right)$ patterns.

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:

$$\begin{aligned} f^{W_s}(\beta) &= \sum_{B_p \in V_G} f^p(\beta) + \sum_{B_q \in W_s \setminus V_G} f^q(\beta) \\ &< \sum_{B_p \in V_G} f^p(\beta) + \sum_{B_q \in W_s \setminus V_G} (f^k_q - \varphi\Delta) \end{aligned} \tag{14}$$

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:

$$\begin{aligned} f^p(\beta) &\leq f^{\widehat{q(p)}}(\beta) + |p - \widehat{q}(p)|\Delta \\ &< f^k_{\widehat{q(p)}} - \varphi\Delta + |p - \widehat{q}(p)|\Delta \\ &< f^k_p - \varphi\Delta + 2|p - \widehat{q}(p)|\Delta \end{aligned} \tag{15}$$

where the second inequality holds because β is below the band gap in $B_{\widehat{q(p)}}$ and (15) follows from *Lemma 1*. Using (15) in (14) we get

$$\begin{aligned} f^{W_s}(\beta) &< \sum_{B_{s'} \in W_s} f^k_{s'} - m\varphi\Delta + \sum_{B_p \in V_G} 2|p - \widehat{q}(p)|\Delta \\ &< \sum_{B_{s'} \in W_s} f^k_{s'} - m\varphi\Delta + 2(1 + 2 + \cdots + g)\Delta \\ &= \sum_{B_{s'} \in W_s} f^k_{s'} - m\varphi\Delta + g(g+1)\Delta = \text{UB} \end{aligned} \tag{16}$$

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 $\mu = \min\{m - v + 1, \frac{\varphi}{2}, \frac{1}{2}(\sqrt{1 + 2m\varphi} - 1)\}$. $\square$

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 patterns. 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 episodes. 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 pattern 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 (Patnaik et al., 2012)).

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 patterns over W_s. If we have $|\mathcal{P}| \geq k$, then the maximum error-rate in the top-k patterns 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 $\frac{\varphi}{2} > 1$
3. $\left(\frac{4\epsilon k m^2}{m^2-1}\right)$ for $v=\left\lfloor\frac{m+1}{2}\right\rfloor$, if $\frac{\varphi}{2} > \frac{1}{m}\left\lceil\frac{m-1}{2}\right\rceil\left\lceil\frac{m+1}{2}\right\rceil$

Proof. We show the proof only for $v=\left\lfloor\frac{m+1}{2}\right\rfloor$. The cases of $v=1$ and $v=m$ are obtained immediately upon application of *Theorem 3*.

Fixing $v=\left\lfloor\frac{m+1}{2}\right\rfloor$ implies $(m-v)=\left\lceil\frac{m-1}{2}\right\rceil$. For $m \geq 2$, $\frac{\varphi}{2} > \frac{1}{m}\left\lceil\frac{m-1}{2}\right\rceil\left\lceil\frac{m+1}{2}\right\rceil$ implies $\frac{\varphi}{2} > \max\{1,(1-\frac{v}{m})(m-v+1)\}$. Let $t_{\min} = \min\{m-v+1, \frac{\varphi}{2}\}$. The minimum value of $t_{\min}$ is governed by

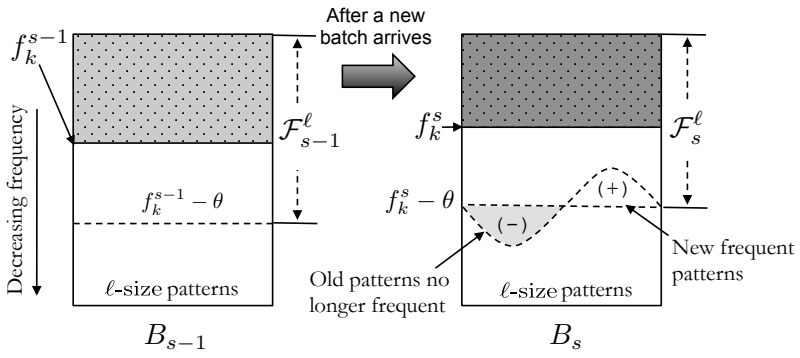
$$\begin{aligned} t_{\min} &\geq \min\left\{\left\lceil\frac{m+1}{2}\right\rceil, \frac{1}{m}\left\lceil\frac{m-1}{2}\right\rceil\left\lceil\frac{m+1}{2}\right\rceil\right\} \\ &= \frac{1}{m}\left\lceil\frac{m-1}{2}\right\rceil\left\lceil\frac{m+1}{2}\right\rceil \\ &\geq \left(\frac{m^2-1}{4m}\right) \end{aligned} \tag{17}$$

Let $g_{\min} = \frac{1}{2}(\sqrt{1+2m\varphi}-1)$. $\varphi > \frac{2}{m}\left\lceil\frac{m-1}{2}\right\rceil\left\lceil\frac{m+1}{2}\right\rceil$ implies $g_{\min} > \left(\frac{m-1}{2}\right)$. From *Theorem 3* we have

$$\mu = \min\{t_{\min}, g_{\min}\} \geq \left(\frac{m^2-1}{4m}\right)$$

and hence the number of errors is no more than $\left(\frac{4\epsilon k m^2}{m^2-1}\right)$. □

Using $v=1$ we make roughly ϵk errors by considering only persistent patterns 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



[t]

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

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 $v = \lceil \frac{m+1}{2} \rceil$ lies in-between, with roughly $4\epsilon k$ errors under reasonable separation conditions.

5. Algorithm

In this section we present an efficient algorithm for incrementally mining patterns 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 patterns (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 patterns with frequency greater than $(f_k^s - \theta)$ in B_s is as follows: Apply an Apriori-based pattern mining algorithm (e.g. (Mannila et al., 1997; Achar et al., 2012)) on batch B_s. If the number of patterns of size-ℓ is less than k, the support threshold is decreased and the mining repeated until at least k ℓ-size patterns 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 patterns that were frequent in the previous batch. This can be exploited to incrementally and efficiently update the set of frequent patterns in the new batch. The intuition is that the frequencies of most patterns do not change much from one batch to the next. As a result only a small number of previously frequent patterns fall below the new support threshold in the new batch; similarly, some new patterns may become frequent. This is illustrated in Figure 3. In order to efficiently find these sets of patterns, 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 patterns are discovered level-wise, in ascending order of pattern-size. An Apriori procedure alternates between counting and candidate generation. First a set $\mathcal{C}^i$ of candidate i-size patterns is determined by combining frequent $(i-1)$-size patterns from the previous level. Then the data is scanned for deter-

mining frequencies of the candidates, from which the frequent i-size patterns are obtained. We note that all candidate patterns that are not frequent constitute the negative border of the frequent pattern lattice (Aumann et al., 1999). This is because, a candidate is generated only when all its subpatterns are frequent. The usual approach is to discard the border. For our purposes, the border contains information required to identify changes in the frequent patterns from one batch to the next[3].

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

For the first batch B_1 (lines 1–3) the top-k patterns are found by a brute-force method, i.e., by mining with a progressively lower frequency threshold until at least k patterns of size ℓ are found. Once f^1_k 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^s_k. Based on *Remark 1*, if $\theta \geq 2\Delta$, $\mathcal{F}^\ell_{s-1}$ from batch B_{s-1} contains every potential top-k pattern of the next batch B_s. Therefore simply updating the counts of all patterns in $\mathcal{F}^\ell_{s-1}$ in the new batch B_s and picking the k^{th} highest frequency gives f^s_k (lines 4–6). To update the lattice of frequent and border patterns (lines 7–24) the procedure starts from the bottom (size-1 patterns). The data is scanned to determine the frequency of new candidates together with the frequent and border patterns from the lattice (line 11). In the first level (patterns of size 1), the candidate set is empty. After counting, the patterns from $\mathcal{F}^\ell_{s-1}$ that continue to be frequent in the new batch are added to $\mathcal{F}^\ell_s$ (lines 13–14). But if a pattern is no longer frequent it is marked as a border set and all its super patterns are deleted (lines 15–17). This ensures that only border patterns are retained in the lattice. In the border and new candidate sets, any pattern that is found to be frequent is added to both $\mathcal{F}^i_s$ and $\mathcal{F}^i_{new}$ (lines 18–21). The remaining infrequent patterns belong to the border because, otherwise, they would have at least one infrequent subpattern and would have been deleted at a previous level; hence, these infrequent patterns are added to $\mathcal{B}^\ell_s$ (lines 22–23).

Finally, the candidate generation step (line 24) is required to fill out the missing parts of the frequent pattern lattice. We want to avoid a full blown candidate generation. Note that if a pattern is frequent in B_{s-1} and B_s then all its subpatterns are also frequent in both B_s and B_{s-1}. Any new pattern ($\notin \mathcal{F}^\ell_{s-1} \cup \mathcal{B}^\ell_{s-1}$) that turns frequent in B_s, therefore, must have at least one subpattern that was not frequent in B_{s-1} but is frequent in B_s. All such patterns are listed in $\mathcal{F}^i_{new}$. Hence the candidate generation step (line 24) for the next level generates only candidates with at least one subpattern $\in \mathcal{F}^i_{new}$. 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 $|F^i_{new}| \ll |\mathcal{F}^i_s|$.

[3] Border sets were employed by (Aumann et al., 1999) 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 pattern miner: Mine top-k v-persistent patterns.

Input: Number k of top patterns; New batch B_s of events;
Current lattice of frequent & border patterns $(\mathcal{F}^*_{s-1}, \mathcal{B}^*_{s-1})$;
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 patterns $(\mathcal{F}^*_s, \mathcal{B}^*_s)$ after B_s

if $s = 1$ **then**
  Determine f^1_k (brute force)
  Compute $(\mathcal{F}^*_1, \mathcal{B}^*_1) \leftarrow$ **Apriori**$(B_1, \ell, f^1_k - \theta)$
else
  CountFrequency$(\mathcal{F}^\ell_{s-1}, B_s)$
  Determine f^s_k (based on patterns in $\mathcal{F}^\ell_{s-1}$)
  Initialize $\mathcal{C}^1 \leftarrow \phi$ (new candidates of size 1)
  for $i \leftarrow 1, \ldots, \ell$ **do**
    Initialize $\mathcal{F}^i_s \leftarrow \phi$, $\mathcal{B}^i_s \leftarrow \phi$
    Initialize $\mathcal{F}^i_{new} \leftarrow \phi$ (new frequent patterns of size i)
    CountFrequency$(\mathcal{F}^i_{s-1} \cup \mathcal{B}^i_{s-1} \cup \mathcal{C}^i, B_s)$
    for $\alpha \in \mathcal{F}^i_{s-1}$ **do**
      if $f^s(\alpha) \geq f^s_k - \theta$ **then**
        Update $\mathcal{F}^i_s \leftarrow \mathcal{F}^i_s \cup \{\alpha\}$
      else
        Update $\mathcal{B}^i_s \leftarrow \mathcal{B}^i_s \cup \{\alpha\}$
        Delete super-patterns of α from $(\mathcal{F}^*_{s-1}, \mathcal{B}^*_{s-1})$
    for $\alpha \in \mathcal{B}^i_{s-1} \cup \mathcal{C}^i$ **do**
      if $f^s(\alpha) \geq f^s_k - \theta$ **then**
        Update $\mathcal{F}^i_s \leftarrow \mathcal{F}^i_s \cup \{\alpha\}$
        Update $\mathcal{F}^i_{new} \leftarrow \mathcal{F}^i_{new} \cup \{\alpha\}$
      else
        Update $\mathcal{B}^i_s \leftarrow \mathcal{B}^i_s \cup \{\alpha\}$
    $\mathcal{C}^{i+1} \leftarrow$ **GenerateCandidate**$(\mathcal{F}^i_{new}, \mathcal{F}^i_s)$
return $(\mathcal{F}^*_s, \mathcal{B}^*_s)$

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 patterns can be obtained by picking the top-k most frequent patterns from the set $\mathcal{F}^\ell_s$. Each pattern 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 patterns are listed in decreasing order of their window counts.

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

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

In the new batch B_s, AB falls out of the frequent set. AB now becomes the new border and all its super-patterns namely ABC, BCD and $ABCD$ are deleted from the lattice.

At level 2, the border pattern 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

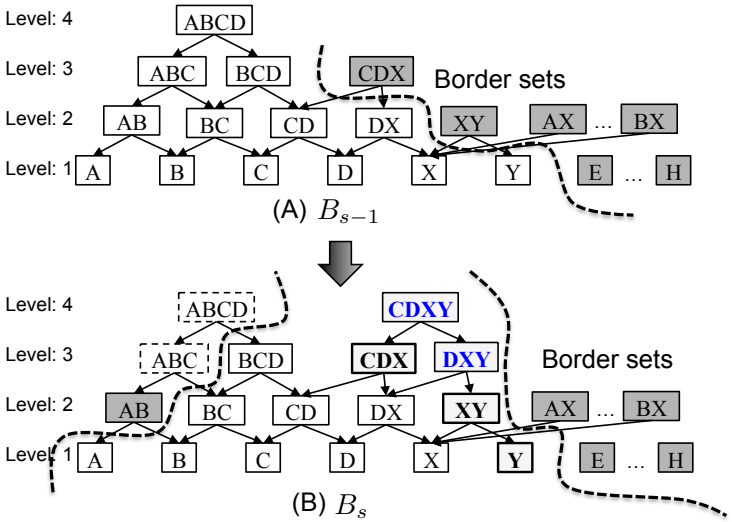


Fig. 4. Incremental lattice update for the next batch B_s given the lattice of frequent and border patterns in B_{s-1}.

$CDXY$ 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 pattern lattice that become frequent in the new data.

5.1. Estimating Δ dynamically

The parameter Δ in the bounded rate change assumption is a critical parameter in the entire formulation. But unfortunately the choice of the correct value for Δ is highly data-dependent. In the streaming setting, the characteristics of the data can change over time. Hence one predetermined value of Δ cannot be provided in any intuitive way. Therefore we estimate Δ from the frequencies of ℓ-size patterns in consecutive windows. We compute the differences in frequencies of patterns that are common in consecutive batches. Specifically, we consider the value at the 75th percentile as an estimate of Δ. We avoid using the maximum change as it tends to be noisy. A few patterns exhibiting large changes in frequency can skew the estimate and adversely affect the mining procedure.

6. Results

6.1. Experimental Setup

We show experimental results for mining two different pattern classes, namely, episodes and itemsets. For episode mining we have 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 (Achar et al., 2012), and

Table 2. Experimental setup.

(a) Datasets used in the experiments.

Dataset	Alphabet-size	Data-length	Type
Call-logs	8	42,030,149	Events
Neuroscience	58	12,070,634	Events
Synthetic	515	25,295,474	Events
Kosarak	41,270	990,000	Itemsets
T10I4D100K	870	100,000	Itemsets

(b) Parameter set-up.

Parameter	Value
k (in Top-k patterns)	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)	$m/2$

(2) real neuronal spiking activity from dissociated cortical cultures, recorded using multi-electrode arrays at Steve Potter's lab, Georgia Tech (Wagenaar et al., 2006). 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. In case of itemsets we present results on two publicly available datasets: Kosarak and T10I4D100K. The Kosarak dataset contains (anonymized) click-stream data of a hungarian on-line news portal while the T10I4D100K dataset was generated using the itemset simulator from the IBM Almaden Quest research group. Both can be downloaded from the Frequent Itemset Mining Dataset Repository `http://fimi.ua.ac.be/data`. The data length in terms of number of events or transactions and the alphabet size of each of these datasets is shown in Table 2(a). Table 2(b) gives the list of parameters and the values used in experiments that we do not explicitly mention in the text.

We compare persistent pattern miner against two methods from itemsets mining literature (Wong and Fu, 2006): 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 algorithm (Manku and Motwani, 2002). These methods were designed for frequent itemset mining and therefore we adapt them suitably for mining episodes. We modify the support threshold computation using chernoff bound for episodes since the total number of distinct episodes is bounded by the size of the largest transaction while for itemsets it is the alphabet size that determines this. As mentioned earlier, we are not aware of any streaming algorithms that directly address top-k *episode* mining over sliding windows of data.

Estimating Δ *dynamically*: Δ is a critical parameter for our persistent pattern 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 patterns in consecutive batches by computing the

change in frequency of patterns and using the 75^{th} percentile as the estimate. We avoid using the maximum change as it tends to be noisy.

6.2. Quality of top-k mining

We present aggregate comparisons of the three competing methods in Table 3. These datasets provide different levels of difficulty for the mining algorithms.

Mining episodes: Tables 3(a) & 3(c) shows high f-scores[4] 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 3(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 3(c) the characteristics are very similar to that of the neuroscience dataset.

Mining itemsets: Tables 3(d) and 3(e) show that for itemset data all the three competing methods perform identically with f-score = 100. But run-times of Persistent pattern mining is significantly better than the other methods with nearly the same or less memory requirement.

6.3. Computation efficiency comparisons

Table 3 shows that we do better than both competing methods in most cases (and never significantly worse than either) with respect to time and memory.

6.3.1. 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.

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

Table 3. Aggregate performance comparison of different algorithms.

(a) Dataset: Neuroscience, Size of patterns = 4, Pattern class: Episodes

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 pattern	97.23	217.64	64.51

(b) Dataset: Call-logs, Size of patterns = 6, Pattern class: Episodes

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 pattern	49.47	3.34	67.7

(c) Dataset: Synthetic data, Size of patterns = 4, Pattern class: Episodes

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 pattern	96.2	4.98	34.43

(d) Dataset: Kosarak, Size of patterns = 4, batch-size = 10,000 transactions, $v = 9$, $m = 10$, Pattern class: Itemsets

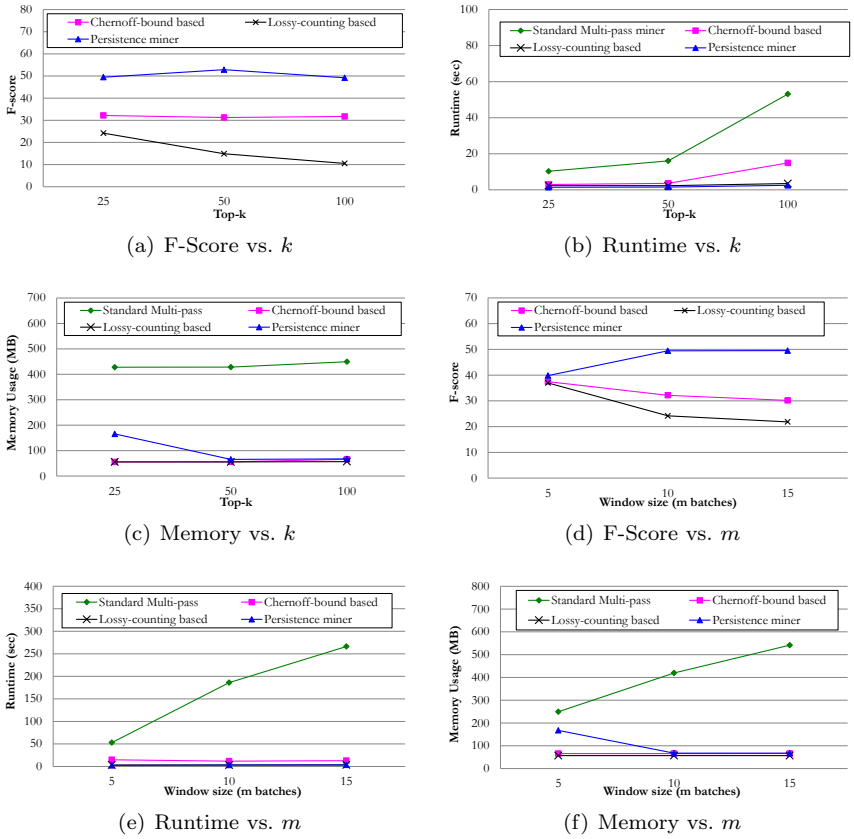
Top-k Miner	F Score	Runtime (sec)	Memory (MB)
Chernoff-bound based	100	32.18	31.85
Lossy-counting based	100	15.57	25.87
Persistent pattern	100	1.93	25.54

(e) Dataset: T10I4D100K, Size of patterns = 4, batch-size = 10,000 transactions, $k = 10$, $v = 9$, $m = 10$, Pattern class: Itemsets

Top-k Miner	F Score	Runtime (sec)	Memory (MB)
Chernoff-bound based	100	662.05	170.77
Lossy-counting based	100	335.71	87.86
Persistent pattern	100	73.38	116.27

6.3.2. Utility of Border Sets

For patterns with slowly changing frequency we show in Section 5 that using border-sets to incrementally update the frequent patterns 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 4 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 pattern lattice from scratch in every new batch. The same batch-wise frequency thresholds used as dictated by Theorem 2. We ran the experiment on our call-logs dataset and *T10I4D100K* dataset, and for various parameter settings of our algorithm we observed a speed-up of $\approx 4\times$ resulting from use of border-sets.



(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)

6.4. 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 (Achar et al., 2012). The simulation model was adapted to update the connection strengths dymanically while generating synthetic data. This allowed us to change the top-k episode patterns slowly over the length of simulated data. We embedded 25 randomly picked episodes with time-varying arrival rates. The arrival rate of each episode was changed by changing the connection strength of between consecutive event types in the episode from 0.1 to 0.9 as ramp function time. By setting a different periodicity of the ramp functions for each episode we were able to change the top-k frequent episodes over time. 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 patterns 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 compet-

Table 4. Utility of border set.
(a) Dataset: Call-logs, Size of patterns = 6, Parameter $v = m/2$, Pattern class: Episodes

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 patterns = 6, window size $m = 10$, Patther class: Episodes

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

(c) Dataset: T10I4D100K, Size of patterns = 4, window size $m = 10$, Pattern class: Itemsets

Parameter v	Runtime (border-set)	Runtime (no-border-set)	Memory (border-set)	Memory (no-border-set)
9	73.38	285.46	116.27	79.89

ing methods (see Figure 6(b). Lossy counting based methods is the slowest at error parameter set to 0.0001. The effective batch-wise thresholds of both the lossy counting and Chernoff bounds based algorithm were very similar leading to identical performance of the two competing methods in this experiments.

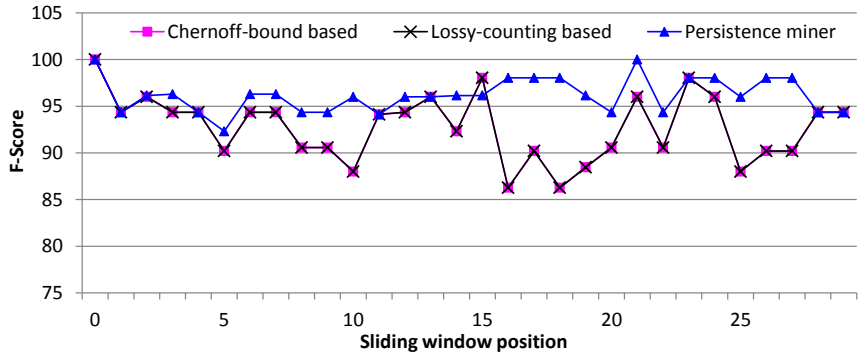
The main reason of better tracking in the case of Persistence miner is that the output of the algorithm filters out all non-(v, k) persistent patterns. This acts in favor of Persistence miner as the patterns most likely to gather sufficient support to be in the top-k are also likely to be persistent.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

6.5. Correlation of f-score with theoretical error

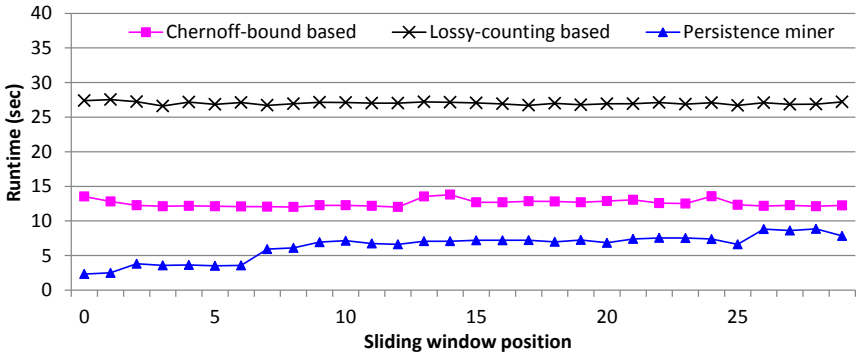
Can we compute theoretical errors (data-dependent quantity) and guess how well we perform? This could be invaluable in a real-world streaming data setting.

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 allowable 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 predicted error bound i.e. the predicted error for



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



(b) Tracking top-k - runtime

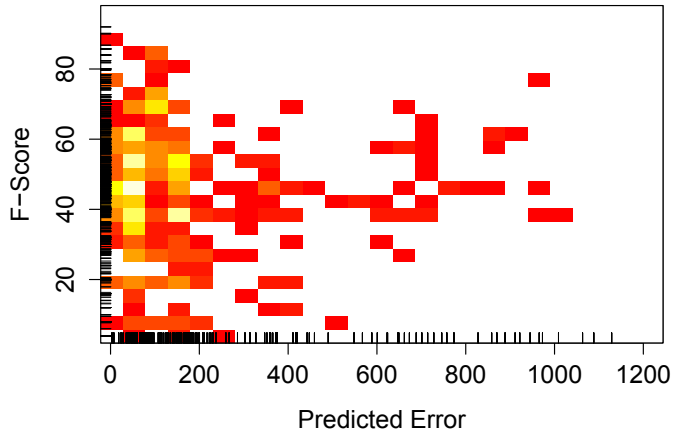
Fig. 6. Comparison of different methods in tracking top-k patterns over dynamically changing event stream (Parameters used: k = 25, m = 10, Persistence miner: v=0.5, alg 6,7: error = 1.0e-4)

high-f-score top-k results should be low and vice 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.

7. Related work

Traditionally, *data stream management systems* (*DSMSs*) (Babcock et al., 2002) have been used mainly for detecting patterns and conditions mined over offline data using traditional multi-pass frequent patterns, frequent itemsets, and sequential patterns and motif-detection techniques. However, as real-time data acquisition becomes ubiquitous, there is a growing need to derive insights directly from high-volume streams at low latencies and under limited memory using a DSMS (Chandramouli et al., 2012).

Mining Itemsets over Streams The literature for streaming algorithms for pattern discovery is dominated by techniques from the frequent itemset min-



[ht]

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

ing context (Manku and Motwani, 2002; Karp et al., 2003; Chang and Lee, 2003; Chang and Lee, 2004; Jin and Agrawal, 2005; Wong and Fu, 2006; Calders et al., 2007; Cheng et al., 2008; Lam et al., 2011), and we are unaware of any algorithms for pattern mining over event streams that possess the generality of mining approach we have presented here. Manku and Motwani (Manku and Motwani, 2002) 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. (Karp et al., 2003) also propose a one pass streaming algorithm for finding frequent items, and these ideas were extended to itemsets by Jin and Agrawal (Jin and Agrawal, 2005), but all these methods require even more space than lossy counting. Mendes et al. (Mendes, Ding and Han, 2008) extend the pattern growth algorithm (PrefixSpan) (Pei et al., 2001) for mining sequential patterns to incorporate the idea of lossy counting. Chang and Lee (Chang and Lee, 2004) and Wong and Fu (Wong and Fu, 2006) 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. (Calders et al., 2007) Lam et al.(Lam et al., 2011)) but these methods are heavily specialized toward the itemset context. There has also been renewed recent interest in adapting pattern matching (Chandramouli et al., 2010; Agrawal et al., 2008) (as opposed to mining) algorithms to the streaming context. It is not obvious how to extend them to accommodate patterns in a manner that supports both candidate generation and counting.

Mining Time-Series Motifs An online algorithm for mining time series motifs is proposed by Mueen et al. (Mueen and Keogh, 2010). The algorithm uses an interesting data structure to find a pair of approximately repeating subsequences in a window. The Euclidean distance measure is used to measure the similarity of the motif sequences in the window. Unfortunately this notion does not extend naturally to discrete patterns. Further, this motif mining formulation does not

explicitly make use of a support or frequency threshold and returns exactly one pair of motifs that are found to be the closest in terms of distance.

Sequential Patterns and pattern Mining An *pattern* or a general partial order pattern can be thought of as a generalization of itemsets where each item in the set is not confined to occur within the same transaction (i.e., at the same time tick) and there is additional structure in the form of ordering of events or items. In serial patterns, events must occur in exactly one particular order. Partial order patterns allow multiple orderings. In addition there could be repeated event types in a pattern. The loosely coupled structure of events in a pattern results in narrower separation between the frequencies of true and noisy patterns (i.e., resulting from random co-occurrences of events) and quickly leads to combinatorial explosion of candidates when mining at low frequency thresholds. Most of the itemset literature does not deal with the problem of candidate generation. The focus is on counting and not so much on efficient candidate generation schemes. While pattern mining in offline multi-pass scenarios has been researched (Laxman, 2006; Achar et al., 2012), this paper is the first to explore ways of doing both counting and candidate generation efficiently in a streaming setting. We devise algorithms that can operate at as high frequency thresholds as possible and yet give certain guarantees about frequent output patterns.

8. Conclusions

While pattern mining in offline multi-pass scenarios has been researched (Agrawal and Srikant, 1994; Mannila et al., 1997; Achar et al., 2012), this paper is the first to explore ways of doing both counting and candidate generation efficiently in a streaming setting. The key ingredients of our approach involve parameterization of the streaming algorithm using data-driven quantities such as rate of change and top-k separation. This leads to algorithms that can operate at high frequency thresholds and with theoretical guarantees on the quality of top-k approximation. Further, our algorithms employ border sets in a novel way, ensuring minimal computation redundancy as we process adjacent (potentially similar) batches. Experiments demonstrate the effectiveness of our algorithms for two popular pattern classes, namely, itemsets and episodes. The approach presented here is applicable for any general pattern class provided we have black-box access to a level-wise algorithm for that class.

Acknowledgements

This work is supported in part by US National Science Foundation grants IIS-0905313 and CCF-0937133, and by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior National Business Center (DoI/NBC) contract number D12PC000337. The US Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoI/NBC, or the US Government.

References

Achar, A. et al. (2012), 'Discovering injective episodes with general partial orders', *Data Mining and Knowl. Dsc.* **25**(1), 67–108.

Agrawal, J. et al. (2008), Efficient pattern matching over event streams, *in* 'Proc. of the 2008 ACM SIGMOD Intl. Conf. on Management of Data', SIGMOD '08, pp. 147–160.

Agrawal, R. and Srikant, R. (1994), Fast algorithms for mining association rules in large databases, *in* 'Proceedings of the 20th International Conference on Very Large Data Bases', pp. 487–499.

Aumann, Y. et al. (1999), 'Borders: An efficient algorithm for association generation in dynamic databases', *J. of Intl. Info. Syst. (JIIS)* (1), 61–73.

Babcock, B. et al. (2002), Models and issues in data stream systems, *in* 'Proc. of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems', pp. 1–16.

Calders, T. et al. (2007), Mining frequent itemsets in a stream, *in* 'Proc. of the 7th IEEE Intl. Conf. on Data Mining (ICDM)', pp. 83–92.

Chandramouli, B. et al. (2010), 'High-performance dynamic pattern matching over disordered streams', *Proc. VLDB Endow.* **3**(1-2), 220–231.

Chandramouli, B. et al. (2012), Temporal analytics on big data for web advertising, *in* 'Proc. of the Intl. Conf. of Data Engg. (ICDE)'.

Chang, J. H. and Lee, W. S. (2003), 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)', pp. 487–492.

Chang, J. H. and Lee, W. S. (2004), 'A sliding window method for finding recently frequent itemsets over online data streams', *J. Inf. Sci. Eng.* **20**(4), 753–762.

Cheng, J. et al. (2008), 'A survey on algorithms for mining frequent itemsets over data streams', *Knowl. and Info. Systems* **16**(1), 1–27.

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

Karp, R. M. et al. (2003), 'A simple algorithm for finding frequent elements in streams and bags', *ACM Trans. Database Syst.* **28**, 51–55.

Lam, H. T. et al. (2011), Online discovery of top-k similar motifs in time series data, *in* 'SIAM Intl. Conf. of Data mining', pp. 1004–1015.

Laxman, S. (2006), Discovering frequent episodes: Fast algorithms, Connections with HMMs and generalizations, PhD thesis, IISc, Bangalore, India.

Laxman, S. and Sastry, P. S. (2006), 'A survey of temporal data mining', *SĀDHĀNĀ, Academy Proceedings in Engineering Sciences* **31**, 173–198.

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

Mannila, H. et al. (1997), 'Discovery of frequent episodes in event sequences', *Data Mining and Knowl. Dsc.* **1**(3), 259–289.

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

Mueen, A. and Keogh, E. (2010), Online discovery and maintenance of time series motif, *in* 'Proc. of the 16th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining (KDD)'.

Muthukrishnan, S. (2005), 'Data streams: Algorithms and applications', *Foundations and Trends in Theoretical Computer Science* **1**(2).

Patnaik, D., Laxman, S. and Ramakrishnan, N. (2009), 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)'.

Patnaik, D., Marwah, M., Sharma, R. and Ramakrishnan, N. (2009), Sustainable operation and management of data center chillers using temporal data mining, *in* 'Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining', pp. 1305–1314.

Patnaik, D. et al. (2012), 'Streaming algorithms for pattern discovery over dynamically changing event sequences', *CoRR* **abs/1205.4477**.

Pei, J. et al. (2001), Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth, *in* 'Proc. of the 17th Intl. Conf. on Data Engg. (ICDE)', pp. 215–224.

Ramakrishnan, N., Patnaik, D. and Sreedharan, V. (2009), 'Temporal Process Discovery in Many Guises', *IEEE Computer* **Vol. 42**(8), 97–101.

Wagenaar, D. A. et al. (2006), 'An extremely rich repertoire of bursting patterns during the development of cortical cultures', *BMC Neuroscience* .

Wang, J. et al. (2005), 'TFP: An efficient algorithm for mining top-k frequent closed itemsets', *IEEE Trans. on Knowledge and Data Engg.* **17**(5), 652–664.

Wong, R. C.-W. and Fu, A. W.-C. (2006), 'Mining top-k frequent itemsets from data streams', *Data Mining and Knowl. Dsc.* **13**, 193–217.

Yan, X. and Han, J. (2003), CloseGraph: mining closed frequent subgraph patterns, *in* 'Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'03)'.

Author Biographies

Debprakash Patnaik is currently an engineer in the Search and Discovery group at Amazon.com. He completed his Ph.D. in Computer Science from Virginia Tech, USA. He received his bachelor degree from Veer Surendra Sai University of Technology, Orissa, India, in 2002 and masters degree from Indian Institute of Science, Bangalore, in 2006. He also worked as a researcher in the Diagnosis and Prognosis group at General Motors Research, Bangalore. His research interests include temporal data mining, probabilistic graphical models, and application in neuroscience.

Srivatsan Laxman Srivatsan Laxman is a researcher at Microsoft Research India, Bangalore and is associated with two research areas, Machine Learning & Optimization and Security & Privacy. He works broadly in data mining, machine learning and data privacy. Srivatsan received his PhD in Electrical Engineering in 2006 from Indian Institute of Science, Bangalore.

Badrish Chandramouli Badrish Chandramouli is a researcher in the database group at Microsoft Research. He is broadly interested in databases and distributed systems, with particular interests in stream processing and temporal analytics, Cloud computing, database query processing and optimization, and publish/subscribe systems. Badrish works primarily on the streams project at Microsoft Research, Redmond, which ships commercially as Microsoft StreamInsight.

Naren Ramakrishnan Naren Ramakrishnan is the Thomas L. Phillips professor of engineering at Virginia Tech. He works with domain scientists and engineers to develop software systems for scientific knowledge discovery. Application domains in his research have spanned protein design, biochemical networks, neuro-informatics, plant physiology, wireless communications, and sustainable design of data centers. Ramakrishnan received his Ph.D. in computer sciences from Purdue University. He is an ACM Distinguished Scientist.

Correspondence and offprint requests to: Debprakash Patnaik, Amazon.com, WA 98109, USA.
Email: patnaikd@amazon.com