Maintaining Frequent Itemsets over High-Speed Data Streams[*]

James Cheng, Yiping Ke, and Wilfred Ng

Department of Computer Science
Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong, China
{csjames, keyiping, wilfred}@cs.ust.hk

Abstract. In this paper, we propose a false-negative approach to approximate the set of frequent itemsets over a sliding window. Existing approximate algorithms use an error parameter, ϵ, to control the accuracy of the mining result. However, the use of ϵ leads to a dilemma. The smaller the value of ϵ, the more accurate is the mining result but the higher the computational complexity, while increasing ϵ degrades the mining accuracy. We address this dilemma by introducing a progressively increasing minimum support function. When an itemset is retained in the window longer, we require its minimum support to approach the minimum support of a frequent itemset. Thus, the number of potential frequent itemsets to be maintained is greatly reduced. Our experiments show that our algorithm not only attains highly accurate mining results, but also runs significantly faster and consumes less memory than do existing algorithms for mining frequent itemsets over a sliding window.

1 Introduction

Frequent itemset (*FI*) mining [1] is fundamental to many important data mining tasks such as associations and correlations. Recently, the increasing prominence of data streams has led to the study of online mining of FIs, which is an important technique to a wide range of applications [7], such as web log and click-stream mining, network traffic analysis, trend analysis and fraud/anomaly detection in telecom data, e-business and stock market analysis, and sensor networks. With the rapid emergence of these new application domains, it has become increasingly demanding to conduct advanced analysis and data mining over data streams to capture interesting trends, patterns and exceptions.

Unlike mining on static datasets, mining data streams poses many new challenges. First, it is unrealistic to keep the entire stream in main memory or even in secondary storage, since a data stream comes continuously and the amount of data is unbounded. Second, traditional methods of mining on stored datasets by multiple scans are infeasible since the streaming data is passed only once.

[*] This work is partially supported by RGC CERG under grant number HKUST6185/02E and HKUST6185/03E.

Third, mining streams requires fast, real-time processing in order to keep up with the high data arrival rate and mining results are expected to be available within short response times. In addition to the unbounded memory requirement and the high arrival rate of a stream, the combinatorial explosion of itemsets exacerbates mining FIs over streams in terms of both memory consumption and processing efficiency. Due to these constraints, research studies have been conducted on approximating mining results.

Existing approximation techniques for mining FIs are mainly *false-positive*[1] [9, 13, 3, 12, 4, 8, 5]. Most of these approaches use an *error parameter*, ϵ, to control the quality of the approximation. However, the use of ϵ leads to a dilemma. A smaller ϵ gives a more accurate mining result. Unfortunately, a smaller ϵ also results in an enormously larger number of itemsets to be maintained, thereby drastically increasing the memory consumption and lowering processing efficiency. A *false-negative*[2] approach [15] is proposed recently to address this dilemma. However, the method focuses on the entire history of a stream and does not distinguish recent itemsets from old ones.

In this paper, we propose a false-negative approach to mine FIs over high-speed data streams. Our method places greater importance on recent data by adopting a sliding window model. To tackle the problem introduced by the use of ϵ, we consider ϵ as a *relaxed minimum support threshold* and propose to progressively increase the value of ϵ for an itemset as it is kept longer in a window. In this way, the number of itemsets to be maintained is greatly reduced, thereby saving both memory and processing power. We design a progressively increasing minimum support function and devise an algorithm to mine FIs over a sliding window. Our experiments show that our approach is able to obtain highly accurate mining results even with a large ϵ, so that the mining efficiency is significantly improved. In most cases, our algorithm runs significantly faster and consumes less memory than do the state-of-the-art algorithms [13, 5], while attains the same level of accuracy.

Organization. We discuss the related work in Section 2 and give the background in Section 3. In Sections 4 and 5, we introduce the progressively increasing minimum support function and present our algorithm. We analyze the quality of the approximation of our approach in Section 6. We present our experimental results in Section 7 and conclude the paper in Section 8.

2 Related Work

Existing streaming algorithms on mining FIs mainly focus on a landmark window [9, 13, 12, 15]. However, these approaches do not distinguish recent itemsets from old ones. Since the importance of an itemset in a stream usually decreases with

[1] The false-positive approach returns a set of itemsets that includes all FIs but also some infrequent itemsets.

[2] The false-negative approach returns a set of itemsets that does not include any infrequent itemsets but misses some FIs.

time, methods that discount the importance of old itemsets exponentially with time have been proposed [3, 8].

Another well-known approach to place greater importance on recent data is adopting the sliding window model [11, 4–6]. Lee et al. [11] propose to mine the exact set of FIs. Their method needs to scan the *entire* window and computes the FIs from the candidate 2-itemsets for each slide. This method is expensive, especially when the window is large, and is thus more suitable for offline processing. Chang and Lee [4, 5] adopt the estimation mechanism of the *Carma* algorithm [9] and that of the *Lossy Counting* algorithm [13], respectively, to mine an approximate set of FIs. The two approaches incrementally update a set of itemsets that are potential to be frequent for each incoming and expiring transaction. We implement their algorithm [5] that adopts the *Lossy Counting* estimation mechanism and a variant of the same algorithm that performs the update for each batch of transactions instead of for each transaction. We find that the former (update-per-transaction) is much slower and consumes much more memory than the latter (update-per-batch), while our approach significantly outperforms the latter as shown by our experimental results. Another work on mining over a sliding window is the *Moment* algorithm proposed by Chi et al. [6]. Their method is not comparable to ours since Moment mines closed FIs [14] instead of FIs.

Yu et al. [15] adopt the *Chernoff bound* to develop a false-negative approach that can control the bound of memory usage and the quality of the approximation by a user-specified probability parameter. Since Chernoff bound requires the size of the stream to become sufficiently large in order to obtain an accurate mining result, it is inflexible to apply it into the sliding window model.

3 Preliminaries

Let $\mathcal{I} = \{x_1, x_2, \ldots, x_m\}$ be a set of items. An *itemset* (or a *pattern*) is a subset of $\mathcal{I}$. A *transaction*, X, is an itemset and X *supports* an itemset, Y, if $X \supseteq Y$. For brevity, we write an itemset $\{x_{j_1}, x_{j_2}, \ldots, x_{j_n}\}$ as $x_{j_1}x_{j_2}\ldots x_{j_n}$.

A *transaction data stream* is a continuous sequence of transactions. We denote a *time unit* in the stream as t_i, within which a variable number of transactions may arrive. A *window* or a *time interval* in the stream is a set of successive time units, denoted as $T = \langle t_i, \ldots, t_j \rangle$, where $i \leq j$, or simply $T = t_i$ if $i = j$. A *sliding window* in the stream is a window that slides forward for every time unit. The window at each slide has a fixed number, w, of time units and w is called the *size* of the window. In this paper, we use t_τ to denote the *current time unit.* Thus, the *current window* is $W = \langle t_{\tau-w+1}, \ldots, t_\tau \rangle$.

We define *trans*(T) as the set of transactions that arrive on the stream in a time interval T and $|trans(T)|$ as the number of transactions in *trans*(T). The *support*[3] of an itemset X over T, denoted as *sup*(X, T), is the number of transactions in *trans*(T) that support X. Given a predefined *Minimum Support Threshold (MST)*, σ $(0 \leq \sigma \leq 1)$, we say that X is a *frequent itemset* (it FI) over T if $sup(X, T) \geq \sigma|trans(T)|$.

[3] The *support* here is the *absolute* occurrence frequency instead of the *relative support.*

t_1	t_2	t_3
abc	bc	bde
bce	abd	ac
bcd		bcd

Table 1. Transactions in Two Windows

Given a transaction data stream and an MST σ, the problem of *FI mining over a sliding window in the stream* is *to find the set of all FIs over the window at each slide.*

Example 1. Table 1 records the transactions that arrive in the stream in two successive windows, $W_1 = \langle t_1, t_2 \rangle$ and $W_2 = \langle t_2, t_3 \rangle$. If the minimum support required is 3 (i.e., $\sigma = 3/5$ in both windows), then the set of FIs over W_1 and W_2 are {`b`, `c`, `bc`} and {`b`, `c`, `d`, `bd`}, respectively. For example, `bc` is an FI over W_1 since *sup*(`bc`, W_1) $= 4$; however, `bc` is infrequent over W_2 since its support over W_2 drops to 2 due to its low frequency in t_3. □

4 A Progressively Increasing MST Function

An itemset may be infrequent at some point in a stream but becomes frequent later. Since there are exponentially many infrequent itemsets at any point in a stream, it is infeasible to keep all infrequent itemsets. Suppose we have an itemset X which is discovered to be frequent at time t. Since X is infrequent before t, the support of X in the stream before t is lost. A common approach [13, 12, 5] to estimate X's support before t is to use an *error parameter*, ϵ, where $0 \leq \epsilon \leq \sigma$. X is maintained in the (sliding or landmark) window as long as its support is no less than ϵN, where N is the number of transactions received in the current window. Thus, if X is kept only after t, the support of X before t is at most ϵN. However, the use of ϵ leads to a dilemma. A small ϵ gives an estimated support close to the true support. Unfortunately, a small ϵ also results in a large number of itemsets to be processed and maintained, thereby drastically increasing the memory consumption and lowering processing efficiency. To tackle this problem, we consider ϵ as a *relaxed MST* and propose to *progressively increase* the value of ϵ as an itemset is kept longer in the window.

We use the relaxed MST $\epsilon = r\sigma$, where r $(0 \leq r \leq 1)$ is the *relaxation rate*, to mine the set of FIs over each time unit t in the sliding window. Since all itemsets whose support less than $r\sigma|trans(t)|$ are discarded, we define the *computed support* of the itemsets as follows.

Definition 1 (Computed Support) The *computed support* of an itemset X *over a time unit* t is defined as follows:

$$\widetilde{sup}(X,t) = \begin{cases} 0 & \text{if } sup(X,t) < r\sigma|trans(t)| \\ sup(X,t) & \text{otherwise,} \end{cases}$$

where $sup(X,t)$ is the *true support* of X over t.

The *computed support* of an itemset X over *a time interval* $T = \langle t_j, \ldots, t_l \rangle$ is defined as

$$\widetilde{sup}(X, T) = \sum_{i=j}^{l} \widetilde{sup}(X, t_i).$$

□

Based on the computed support of an itemset, we apply *a progressively increasing MST function* to define *a semi-frequent itemset.*

Definition 2 (Semi-Frequent Itemset) Let $W = \langle t_{\tau-w+1}, \ldots, t_\tau \rangle$ be a window of size w and $T^k = \langle t_{\tau-k+1}, \ldots, t_\tau \rangle$, where $1 \leq k \leq w$, be the most recent k time units in W. We define a *progressively increasing* function

$$minsup(k) = \left\lceil m_k \ \times \ r_k \right\rceil,$$

where $m_k = \sigma \left| trans\left(T^k\right) \right|$ and $r_k = \left(\frac{1-r}{w}\right)\left(k-1\right) + r$.

An itemset X is a *semi-frequent itemset* (*semi-FI*) over W if $\widetilde{sup}(X, T^k) \geq minsup(k)$, where $k = \tau - o + 1$ and t_o is the oldest time unit such that $\widetilde{sup}(X, t_o) > 0$.

□

The first term m_k in the *minsup* function in Definition 2 is the minimum support required for an FI over T^k, while the second term r_k progressively increases the value of the relaxed MST $r\sigma$ at the rate of $((1-r)/w)$ for each older time unit in the window. We keep an itemset X in the window only if its computed support over T^k is no less than $minsup(k)$, where T^k is the time interval starting from the time unit t_o, in which the support of X is computed, up to the current time unit t_τ.

We remark that k is relative to the specific window at each slide and the computed support of a semi-FI X over $\langle t_{\tau-w+1}, \ldots, t_{\tau-k} \rangle$ is considered as unpromising and discarded. When we re-compute the value of k for the window at the next slide, the computed support of X over $\langle t_{\tau-w+1}, \ldots, t_{\tau-k} \rangle$ will be taken as 0.

We illustrate the concept of semi-FIs by the following example.

k	10	9	8	7	6	5	4	3	2	1
$minsup(k)$	182	148	117	90	66	46	30	17	8	2

Time Unit	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}
sup(ab,t_i)	3	1	2	3	2	1	4	7	11	19	21
sup(cd,t_i)	3	11	20	29	11	8	17	28	37	41	39

Table 2. Two Sample Semi-FIs

Example 2. Let $\sigma = 0.01$, $r = 0.1$ and $w = 10$. We assume a uniform input rate of 2,000 transactions in each time unit. The 11 time units constitute to the windows for two slides, $W_1 = \langle t_1, \ldots, t_{10} \rangle$ and $W_2 = \langle t_2, \ldots, t_{11} \rangle$. Table 2 shows the value of $\mathit{minsup}(k)$, for $1 \leq k \leq 10$, and the support of two itemsets, `ab` and `cd`, over each time unit.

We first discuss the method used by the state-of-the-art algorithm *Lossy Counting* [13, 5], which is the baseline of comparison with our approach in our experiments. Lossy Counting keeps an itemset if its overall support is no less than $r\sigma N$, where N is the number of transactions in the current window. Thus, both `ab` and `cd` are retained in both windows since $r\sigma N = 20$, even though the supports of `ab` over some time units are very low and obviously unpromising. Note that in t_2 and t_6, the support of `ab` is 1 but `ab` is still mined. Mining itemsets with minimum support 1 means mining all itemsets, which is an extremely expensive operation in terms of both time and space, since the number of all itemsets is prohibitively large. Although we only need to consider those itemsets that have already been discovered, the number of these itemsets is still very large when σ is small.

With our progressively increasing MST, the supports of `ab` over the first six time units can at most pass $\mathit{minsup}(1)$=2 and are always less than $\mathit{minsup}(2)$=8 over any two consecutive time units. For `ab` to be frequent in W_1 or W_2, its support must increase rapidly from t_7 to t_{11}, which is unlikely given the low supports of `ab` in the past. Thus, the supports of `ab` over the first six time units can be regarded as *unpromising* and discarded. Although the trend in the support of `ab` from t_7 to t_{11} shows that `ab` may become frequent after a few window slides, the first few time units will have expired at that time. Therefore, it is reasonable that *we require the support of* `ab` *to increase progressively as it stays longer in a window.* In this way, `ab` is only retained from t_7 and afterwards, as its supports from t_7 up to t_{11} are greater than the corresponding $\mathit{minsup}(k)$, for $k = 1, \ldots, 5$. Indeed, the supports of `ab` over these time units are more likely to contribute to the overall support of `ab` should it become frequent later.

The supports of `cd` over all the time units, in both W_1 and W_2, are always greater than the corresponding $\mathit{minsup}(k)$, for $k = 1, \ldots, 10$. However, we can see that there are some low supports observed over the first few time units and those in the middle. These low supports are not discarded since they are compensated by the high supports of `cd` over other time units. □

5 Mining FIs over a Sliding Window

We use a prefix tree to keep the semi-FIs of the window at each slide. A node in the prefix tree has three fields:

- *item*: the last item of an itemset[4], X, so that X is represented by the path from the root to the node.
- $\mathit{uid}(X)$: the ID of the time unit, $t_{\mathit{uid}(X)}$, in which X is inserted into the prefix tree.
- $\widetilde{\mathit{sup}}(X)$: the computed support of X since $t_{\mathit{uid}(X)}$.

[4] We assume the items of an itemset are lexicographically ordered.

Algorithm 1 (MineSW)

Input: *(1) An empty prefix tree. (2) σ, r and w. (3) A transaction data stream.*
Output: *An approximate set of FIs of the window at each slide.*

1. We apply an existing non-streaming algorithm to mine all FIs over the set of transactions received within each time unit, using a relaxed MST $r\sigma$.
2. **Initialization:** For each of the first w time units, t_i $(1 \leq i \leq w)$, mine all FIs from $\mathit{trans}(t_i)$. For each mined itemset, X, check if X is in the prefix tree.
   (a) If X is in the prefix tree, perform the following operations.
      i. Add $\widetilde{sup}(X, t_i)$ to $\widetilde{sup}(X)$.
      ii. If $\widetilde{sup}(X) < \mathit{minsup}(i - \mathit{uid}(X) + 1)$, remove X from the prefix tree and stop mining the supersets of X from $\mathit{trans}(t_i)$.
   (b) If X is not in the prefix tree, create a new node for X in the prefix tree with $\mathit{uid}(X) = i$ and $\widetilde{sup}(X) = \widetilde{sup}(X, t_i)$.

   After the itemsets mined from $\mathit{trans}(t_i)$ are updated, traverse the prefix tree once to remove all itemsets X if $\widetilde{sup}(X) < \mathit{minsup}(i - \mathit{uid}(X) + 1)$. When we remove X, we also remove all its descendants since they are its supersets (other supersets of X that are not descendants of X are removed when they are visited in the tree traversal).
3. **Incremental Update:**
   – For each expiring time unit, $t_{\tau-w+1}$, mine all FIs from $\mathit{trans}(t_{\tau-w+1})$. For each mined itemset, X:
      • If X is in the prefix tree and $\tau - \mathit{uid}(X) + 1 \geq w$, subtract $\widetilde{sup}(X, t_{\tau-w+1})$ from $\widetilde{sup}(X)$. Otherwise, stop mining the supersets of X from $\mathit{trans}(t_{\tau-w+1})$.
      • If $\widetilde{sup}(X)$ becomes 0, remove X from the prefix tree. Otherwise, set $\mathit{uid}(X) = \tau - w + 2$.
   – For each incoming time unit, t_τ, mine all FIs from $\mathit{trans}(t_\tau)$. For each mined itemset, X, check if X is in the prefix tree.
      (a) If X is in the prefix tree, perform the following operations.
         i. Add $\widetilde{sup}(X, t_\tau)$ to $\widetilde{sup}(X)$.
         ii. If (1) $\tau - \mathit{uid}(X) + 1 \leq w$ and $\widetilde{sup}(X) < \mathit{minsup}(\tau - \mathit{uid}(X) + 1)$, or (2) $\tau - \mathit{uid}(X) + 1 > w$ and $\widetilde{sup}(X) < \mathit{minsup}(w)$, remove X from the prefix tree and stop mining the supersets of X from $\mathit{trans}(t_\tau)$.
      (b) If X is not in the prefix tree, create a new node for X in the prefix tree with $\mathit{uid}(X) = \tau$ and $\widetilde{sup}(X) = \widetilde{sup}(X, t_\tau)$.
4. **Pruning and Outputting:** Scan the prefix tree once. For each itemset X visited:
   – Remove X and its descendants from the prefix tree if (1) $\tau - \mathit{uid}(X) + 1 \leq w$ and $\widetilde{sup}(X) < \mathit{minsup}(\tau - \mathit{uid}(X) + 1)$, or (2) $\tau - \mathit{uid}(X) + 1 > w$ and $\widetilde{sup}(X) < \mathit{minsup}(w)$.
   – Output X if $\widetilde{sup}(X) \geq \sigma|\mathit{trans}(W)|$ (we can thus set $\mathit{minsup}(w) = \sigma|\mathit{trans}(W)|$ to prune more itemsets).

The algorithm for mining FIs over a sliding window, as described in Algorithm 1, is self-explanatory. In the initialization step, we create the prefix tree for the first window and we make sure all itemsets kept in the prefix tree are semi-frequent. After we finish processing the first w time units, we begin to slide the window.

Before each slide, we first subtract the support of the existing semi-FIs over the first time unit of the window, that is, the expiring time unit $t_{\tau-w+1}$. Note that we need to make sure that the support of an itemset is computed from

$t_{\tau-w+1}$. If $\tau - uid(X) + 1 < w$, then X is inserted after $t_{\tau-w+1}$ and thus we should not decrease the support of X for $t_{\tau-w+1}$. If $\tau - uid(X) + 1 \geq w$, then part of X's support is from $t_{\tau-w+1}$ and needs to be subtracted. The "$\geq$", instead of "$=$", is used because if we continue to keep X in the prefix tree, we set $uid(X)$ to the ID of the next time unit $\tau - w + 2$. Therefore, if X is not mined over the next $i \geq 1$ time units after $t_{\tau-w+1}$, in the coming window $\langle t_{(\tau-w+1)+(i+1)}, \ldots, t_{\tau+i+1} \rangle$ after $(i+1)$ slides, when we compute "$\tau - uid(X) + 1$" again, we have $(\tau + i + 1) - (\tau - w + 2) + 1 = w + i > w$.

After processing the expiring time unit, we process the new time unit t_τ to complete the window at the current slide. This step is similar to processing each time unit in the initialization step, except that the condition for the removal of an itemset X is different. Here, we need to check when X is inserted into the prefix tree so that we can compare $\widetilde{sup}(X)$ with the corresponding *minsup* value. If $\tau - uid(X) + 1 \leq w$, then X is inserted in some time unit within the current window and we check its support against the *minsup* value for the most recent $(\tau - uid(X) + 1)$ time units. If $\tau - uid(X) + 1 > w$ (the "$>$" sign is as explained above), then X was inserted before the current window; thus, we check X's support against $minsup(w)$.

The pruning step following each slide, as well as in initialization, clears those itemsets that no longer satisfy the semi-frequent criterion and are not pruned when we mine the time units (they may not be mined if their subset is not mined). Finally, we output all itemsets whose support is greater than the minimum support. Thus, there is no false-positive.

6 Approximation Quality

In this section, we analyze the quality of the approximation on the mining results returned by our algorithm.

The error bound of the computed support of a semi-frequent itemset X over T^k, $\widetilde{sup}(X, T^k)$, is described as follows:

$$sup(X, T^k) \ - \mathcal{E} \ \ \leq \widetilde{sup}(X, T^k) \ \ \leq \ \ sup(X, T^k),$$

where $\mathcal{E} = \sum_{i:\ \tau-k+1 \leq i \leq \tau \ \wedge \ \widetilde{sup}(X,t_i)=0} \big(r\sigma |trans(t_i)| - 1 \big)$.

Note that $\widetilde{sup}(X, t_i) = 0$ implies that X is infrequent over t_i. Thus, the true support of X over t_i is at most $(r\sigma |trans(t_i)| - 1)$.

Replacing $r\sigma$ with ϵ and $|trans(T^k)|$ with N_k, we obtain an upper bound for the *maximum support error* $\mathcal{E}$ as follows,

$$\mathcal{E} < \epsilon N_k$$

In most cases (except for skewed data distribution which is addressed in Section **??**), the cardinality of the set $\{i : \tau - k + 1 \leq i \leq \tau \ \wedge \ \widetilde{sup}(X, t_i) = 0\}$ is small, since otherwise X would not be semi-frequent. More importantly, the k for the frequent itemsets over each window is mostly equal to w, for which $|\{i : \tau - k + 1 \leq i \leq \tau \ \wedge \ \widetilde{sup}(X, t_i) = 0\}|$ is even smaller since there are no unpromising supports discarded for the first few time units of a window. Thus, in most cases, the maximum support error bound of a frequent itemset is described as follows,

$$\mathcal{E} \ll \epsilon N,$$

where N is the number of transactions over the window.

We remark that ϵN is the maximum support error bound of most existing approximate stream mining algorithms [13, 8, 12, 5].

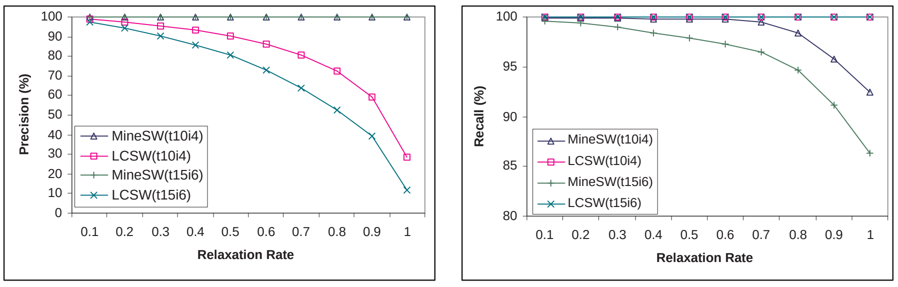
When there are some unpromising supports of an itemset discarded for the first few time units of a window, the maximum support error bound of the itemset can be greater. However, the loss in support over these time units is considered unpromising and it should be noted that these time units are expiring (this is different from the landmark window model where there is no expiring time unit).

Our mining algorithm returns all itemsets whose computed support is at least σN. Thus, there is no false-positives. The set of false-negatives are defined as $\{X \mid \widetilde{sup}(X, W) < \sigma N \ \wedge \ sup(X, W) \geq \sigma N\}$. If $\widetilde{sup}(X, W) < \sigma N$ but $sup(X, W) \geq \sigma N$, then it must be due to some time unit t_i in W such that $\widetilde{sup}(X, t_i) = 0$. Recall that $\widetilde{sup}(X, t_i) = 0$ only if $sup(X, t_i) < \epsilon|trans(t_i)|$ or $\widetilde{sup}(X, t_i)$ is unpromising and discarded. Thus, the true supports of X over other time units in W must be much greater such that we can still have $sup(X, W) \geq \sigma N$. Therefore, we can deduce that false-negatives are mostly itemsets with skewed support distribution over the stream, which can be addressed using a specific *minsup* function as discussed in Section **??**. In most other cases, the number of false-negatives is small as verified by our experiments.

7 Experimental Evaluation

We run our experiments on a Sun Ultra-SPARC III with 900 MHz CPU and 4GB RAM, running Solaris 8 64-bit. We compare our algorithm *MineSW* with a variant of the *Lossy Counting* algorithm [13] applied in the sliding window model, denoted as *LCSW*. We remark that LCSW is different from the algorithm proposed by Chang and Lee [5] as discussed in Section 2. We implement both algorithms and find that the algorithm by Chang and Lee is much slower than LCSW and runs out of memory even with 4GB of RAM.

Datasets: We generate the data streams using the IBM data generator [2, 10] (we are not able to obtain any real dataset that is large enough to model a stream). However, we find that both MineSW and LCSW attain 100% accuracy (MineSW is significantly faster) in most cases, because the distribution of the itemsets in the data streams is too uniform. In order to have a better account on the accuracy of the two approximation techniques, we modify the data generator so that there are new itemsets added and old itemsets expired from the window at each slide. We generate two types of data streams, t10i4 and t15i6, where 10 and 15 (4 and 6) are the average size of a transaction (a maximal frequent itemset) of the two streams, respectively. Each stream consists of 3M transactions and we report the results of MineSW and LCSW averaged over 40 consecutive slides. Each window consists of 20 time units and each time unit receives approximately 50K transactions. At each slide, the window has 3K unique items, while the number of newly-added and old-expiring unique items are limited to 100.



(a) Precision (b) Recall

Fig. 1. Precision and Recall with Varying Relaxation Rate

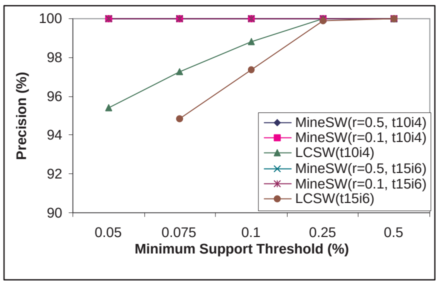
7.1 Varying Relaxation Rate

We first examine the effect of varying the relaxation rate r on MineSW and LCSW (note that $\epsilon = r\sigma$ in LCSW). We set $\sigma = 0.1\%$. Fig. 1 (a) and (b) show that MineSW has 100% precision while LCSW has 100% recall. However, as r increases from 0.1 to 1, the precision of LCSW drops from 98% to around 10%, while the recall of MineSW only drops from 99% to around 90%. The result reveals that the estimation mechanism of the Lossy Counting algorithm relies on ϵ to control the mining accuracy, while our progressively increasing *minsup* function maintains a high accuracy which is only slightly affected by the change in r. Recall that MineSW mines the frequent itemsets over each time unit at an MST $r\sigma$, increasing r means faster mining process and less memory consumption. As a result, this finding enables us to use a larger r to obtain approximately the same mining accuracy but with much faster speed and less memory consumption, as we show in the next section.

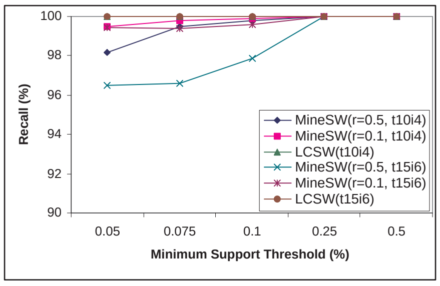
7.2 Varying Minimum Support Threshold

This section assesses the performance of MineSW by varying σ from 0.5% to 0.05%. According to Lossy Counting [13], a good choice of ϵ is 0.1σ. Thus, we set $r = 0.1$ for LCSW. However, the experimental result of last section shows that MineSW can obtain a good accuracy even with a larger r. Thus, we also test $r = 0.5$ for MineSW, in addition to $r = 0.1$ as a baseline. We note that LCSW runs out of memory at $\sigma = 0.05\%$ on the data stream t15i6.

Fig. 2 (a) and (b) show that for both streams and all σ, the precision of LCSW is over 94% and the recall of MineSW is over 96% (mostly over 99%). The recall of MineSW ($r = 0.5$) is only slightly lower than that of MineSW ($r = 0.1$). However, Fig. 3 (a) and (b) show that MineSW ($r = 0.5$) is significantly faster than MineSW ($r = 0.1$). On average, MineSW ($r = 0.5$) is 6.8 times faster than MineSW ($r = 0.1$), while MineSW ($r = 0.1$) is approximately 2 times faster than LCSW. Fig. 4 (a) and (b) show the memory consumption of the algorithms in terms of the number of itemsets maintained at the end of each slide. The number of itemsets kept by MineSW ($r = 0.1$) is about 1.5 times less than that of LCSW, while that kept by MineSW ($r = 0.5$) is less than that of LCSW by up to several orders of magnitude.

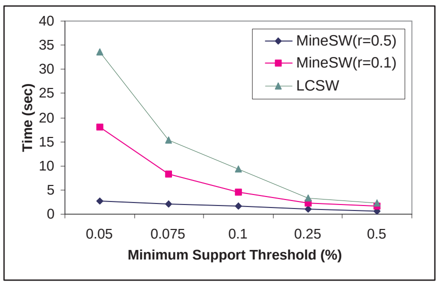


(a) Precision

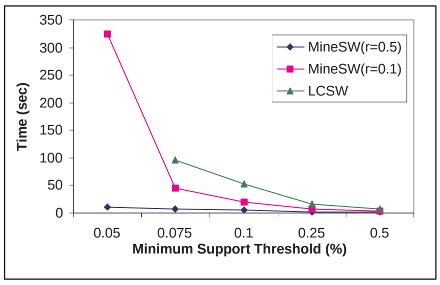


(b) Recall

Fig. 2. Precision and Recall with Varying Minimum Support Threshold

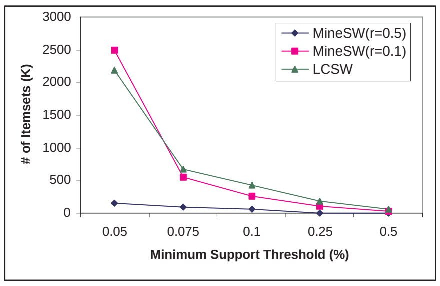


(a) Processing Time (t10i4)

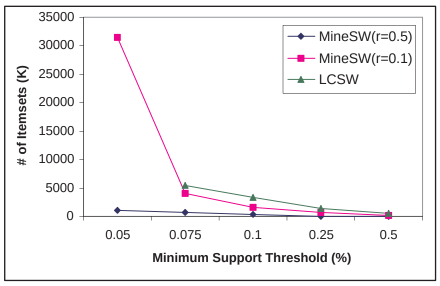


(b) Processing Time (t15i6)

Fig. 3. Processing Time with Varying Minimum Support Threshold



(a) Memory Consumption (t10i4)



(b) Memory Consumption (t15i6)

Fig. 4. Memory Consumption with Varying Minimum Support Threshold

In overall, the performance of MineSW ($r = 0.1$) is about 2 times better than that of LCSW. However, when r is set at 0.5, MineSW not only attains a similar (or better) level of mining accuracy as with LCSW, but also runs faster and consumes less memory than LCSW by an average of over an order of magnitude. Therefore, we can use $r = 0.5$ or even larger values of r to process high-speed data streams and use smaller values of r only when the application demands a very high accuracy (say, using MineSW ($r = 0.1$) to achieve 100% precision and over 99.99% recall).

8 Conclusions

We propose a false-negative approach to mine frequent itemsets over a sliding window. Existing false-positive approaches use an error parameter, ϵ, to control the quality of the approximate mining results. However, a small ϵ results in low mining efficiency while a large ϵ gives a poor mining accuracy. We address this problem by a progressively increasing minimum support function. In our approach, increasing the value of ϵ only slightly and gradually degrades the mining accuracy, but significantly improves the mining efficiency and saves memory usage. We devise an efficient algorithm and verify, by extensive experiments, that our algorithm runs significantly faster and consumes less memory than existing algorithms, but attains the same level of accuracy on the mining results. When applications require highly accurate mining results, our experiments show that by setting $\epsilon = 0.1\sigma$ (a rule-of-thumb choice of ϵ in Lossy Counting [13]), our algorithm attains 100% precision and over 99.99% recall.

References

1. R. Agrawal, T. Imielinski, and A. N. Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proc. of SIGMOD*, 1993.
2. R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules. In *Proc. of VLDB*, 1994.
3. J. H. Chang and W. S. Lee. Finding Recent Frequent Itemsets Adaptively over Online Data Streams. In *Proc. of KDD*, 2003.
4. J. H. Chang and W. S. Lee. estWin: Adaptively Monitoring the Recent Change of Frequent Itemsets over Online Data Streams. In *Proc. of CIKM*, 2003.
5. J. H. Chang and W. S. Lee. A Sliding Window method for Finding Recently Frequent Itemsets over Online Data Streams. In *Journal of Information Science and Engineering*, Vol. 20, No. 4, July, 2004.
6. Y. Chi, H. Wang, P. S. Yu and R. R. Muntz. Moment: Maintaining Closed Frequent Itemsets over a Stream Sliding Window. In *Proc. of ICDM*, 2004.
7. M. Garofalakis, J. Gehrke, R. Rastogi. Querying and Mining Data Streams: You Only Get One Look. In *Tutorial of SIGMOD*, 2002.
8. C. Giannella, J. Han, J. Pei, X. Yan, and P. S. Yu. Mining Frequent Patterns in Data Streams at Multiple Time Granularities. *MIT/AAAI Press*, 2004.
9. C. Hidber. Online Association Rule Mining. In *Proc. of SIGMOD*, 1999.
10. IBM Quest Data Mining Project. The Quest retail transaction data generator. `http://www. almaden.ibm.com/software/quest/`, 1996.
11. C. Lee, C. Lin, and M. Chen. Sliding-window Filtering: an Efficient Algorithm for Incremental Mining. In *Proc. of CIKM*, 2001.
12. H. Li, S. Lee, and M. Shan. An Efficient Algorithm for Mining Frequent Itemsets over the Entire History of Data Streams. In *Proc. of First International Workshop on Knowledge Discovery in Data Streams*, 2004.
13. G. S. Manku and R. Motwani. Approximate Frequency Counts over Data Streams. In *Proc. of VLDB*, 2002.
14. N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering Frequent Closed Itemsets for Association Rules. In *Proc. of ICDT*, 1999.
15. J. Yu, Z. Chong, H. Lu, and A. Zhou. False Positive or False Negative: Mining Frequent Itemsets from High Speed Transactional Data Streams. In *Proc. of VLDB*, 2004.