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. We propose a false-negative approach to approximate the set of *frequent itemsets* (*FIs*) 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. A smaller ϵ gives a more accurate mining result but higher 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 an FI. Thus, the number of potential FIs 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 FIs over a sliding window.

1 Introduction

Frequent itemset (*FI*) mining is fundamental to many important data mining tasks. Recently, the increasing prominence of data streams has led to the study of online mining of FIs [5]. Due to the constraints on both memory consumption and processing efficiency of stream processing, together with the exploratory nature of FI mining, research studies have sought to approximate FIs over streams.

Existing approximation techniques for mining FIs are mainly *false-positive* [5, 4, 1, 2]. 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* approach [6] 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.

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

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

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 obtains 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 [5, 2], while attains the same level of accuracy.

2 Preliminaries

Let $\mathcal{I} = \{x_1, x_2, \ldots, x_m\}$ be a set of items. An *itemset* is a subset of $\mathcal{I}$. A *transaction*, X, is an itemset and X *supports* an itemset, Y, if $X \supseteq Y$. 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* 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* (*FI*) over T if *sup*$(X, T) \geq \sigma|$*trans*$(T)|$.

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

3 A Progressively Increasing MST Function

Existing approaches [5, 4, 2] use an *error parameter*, ϵ, to control the mining accuracy, which leads to a dilemma. We tackle this problem by considering $\epsilon = r\sigma$ as a relaxed MST , 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 is less than $r\sigma|$*trans*$(t)|$ are discarded, we define the *computed support* as follows.

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

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

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

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

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

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

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

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 relaxed MST $r\sigma$ at the rate of $((1-r)/w)$ for each older time unit in the window. We keep 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.

4 Mining FIs over a Sliding Window

We use a prefix tree to keep the semi-FIs. A node in the prefix tree represents an itemset, X, and has three fields: (1) *item* which is the last item of X; (2) $uid(X)$ which is the ID of the time unit, $t_{uid(X)}$, in which X is inserted into the prefix tree; (3) $\widetilde{sup}(X)$ which is the computed support of X since $t_{uid(X)}$.

The algorithm for mining FIs over a sliding window, *MineSW*, is given in Algorithm 1, which is self-explanatory.

Algorithm 1 (MineSW)

```
Input: (1) An empty prefix tree. (2) $\sigma$, $r$ and $w$. (3) A transaction data stream.
Output: An approximate set of FIs of the window at each slide.
1. Mine all FIs over 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 $trans(t_i)$. 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 $\widetilde{sup}(X) < minsup(i - uid(X) + 1)$, remove $X$ from the prefix
       tree and stop mining the supersets of $X$ from $trans(t_i)$.
   (b) If $X$ is not in the prefix tree, create a new node for $X$ in the prefix tree with
       $uid(X) = i$ and $\widetilde{sup}(X) = \widetilde{sup}(X, t_i)$.
3. Incremental Update:
   – For each expiring time unit, $t_{\tau-w+1}$, mine all FIs from $trans(t_{\tau-w+1})$. For each
     mined itemset, $X$:
       • If $X$ is in the prefix tree and $\tau - uid(X) + 1 \geq w$, subtract $\widetilde{sup}(X, t_{\tau-w+1})$
         from $\widetilde{sup}(X)$. Otherwise, stop mining the supersets of $X$ from $trans(t_{\tau-w+1})$.
       • If $\widetilde{sup}(X)$ becomes 0, remove $X$ from the prefix tree. Otherwise, set $uid(X) =$
         $\tau - w + 2$.
   – For each incoming time unit, $t_\tau$, mine all FIs from $trans(t_\tau)$. 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 either $\tau - uid(X) + 1 \leq w$ and $\widetilde{sup}(X) < minsup(\tau -$
         $uid(X) + 1)$, or $\tau - uid(X) + 1 > w$ and $\widetilde{sup}(X) < minsup(w)$, remove $X$
         from the prefix tree and stop mining the supersets of $X$ from $trans(t_\tau)$.
     (b) If $X$ is not in the prefix tree, create a new node for $X$ in the prefix tree
         with $uid(X) = \tau$ 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) $\tau - uid(X) + 1 \leq w$ and $\widetilde{sup}(X) < minsup(\tau - uid(X) + 1)$, or (2) $\tau - uid(X) + 1 > w$ and $\widetilde{sup}(X) < minsup(w)$.
   - Output X if $\widetilde{sup}(X) \geq \sigma|trans(W)|$ (we can thus set $minsup(w) = \sigma|trans(W)|$ to prune more itemsets).

5 Experimental Evaluation

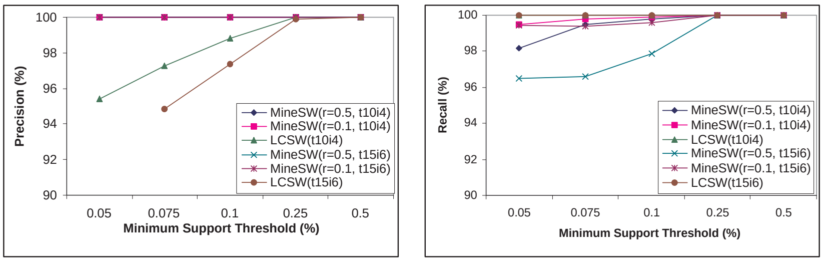
We run our experiments on a Sun Ultra-SPARC III with 900 MHz CPU and 4GB RAM. We compare our algorithm *MineSW* with a variant of the *Lossy Counting* algorithm [5] applied in the sliding window model, denoted as *LCSW*. We remark that LCSW, which updates a batch of incoming/expiring transactions at each window slide, is different from the algorithm proposed by Chang and Lee [2], which updates on each incoming/expiring transaction. We implement both algorithms and find that the algorithm by Chang and Lee is much slower than LCSW and runs out of our 4GB memory. We generate two types of data streams, t10i4 and t15i6, using a generator [3] that modifies the IBM data generator.

We first find (see details in [3]) that when r increases from 0.1 to 1, the precision of LCSW ($\epsilon = r\sigma$ in LCSW) drops from 98% to around 10%, while the recall of MineSW only drops from 99% to around 90%. This 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. Since increasing r means faster mining process and less memory consumption, we can use a larger r to obtain highly accurate mining results at much faster speed and less memory consumption.

We test $r = 0.1$ and $r = 0.5$ for MineSW. According to Lossy Counting [5], a good choice of ϵ is 0.1σ and hence we set $r = 0.1$ for LCSW. Fig. 1 (a) and (b) show that for 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. 2 (a) and (b) show that MineSW ($r = 0.5$) is significantly faster than MineSW ($r = 0.1$) and LCSW, especially when σ is small. Fig. 3 (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.

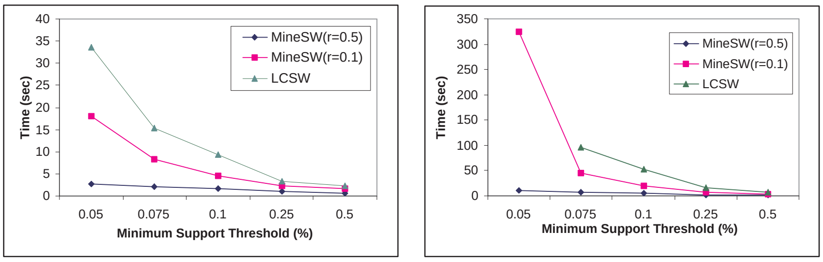
6 Conclusions

We propose a progressively increasing minimum support function, which allows us to increase ϵ at the expense of only slightly degraded accuracy, but significantly improves the mining efficiency and saves memory usage. We verify, by extensive experiments, that our algorithm is significantly faster and consumes less memory than existing algorithms, while attains the same level of accuracy. 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 [5]), our algorithm attains 100% precision and over 99.99% recall.



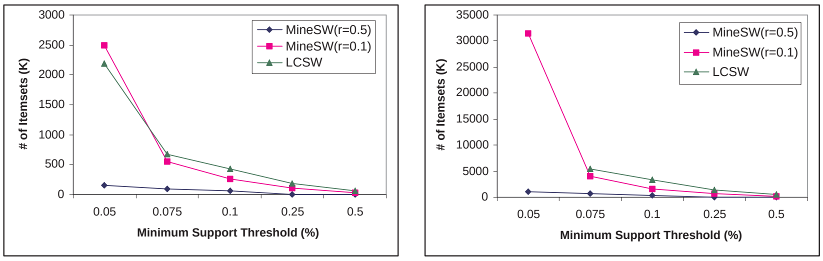
(a) Precision (b) Recall

Fig. 1. Precision and Recall with Varying Minimum Support Threshold



(a) Processing Time (t10i4) (b) Processing Time (t15i6)

Fig. 2. Processing Time with Varying Minimum Support Threshold



(a) Memory Consumption (t10i4) (b) Memory Consumption (t15i6)

Fig. 3. Memory Consumption with Varying Minimum Support Threshold

References

1. 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.
2. 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.
3. J. Cheng, Y. Ke, and W. Ng. Maintaining Frequent Itemsets over High-Speed Data Streams. *Technical Report*, `http://www.cs.ust.hk/∼csjames/pakdd06tr.pdf`.
4. 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.
5. G. S. Manku and R. Motwani. Approximate Frequency Counts over Data Streams. In *Proc. of VLDB*, 2002.
6. J. Yu, Z. Chong, H. Lu, and A. Zhou. False Positive or False Negative: Mining Frequent Itemsets from High Speed Transactional Data Streams. In *VLDB*, 2004.