An Approximate Approach for Mining Recently Frequent Itemsets from Data Streams[*]

Jia-Ling Koh and Shu-Ning Shin

Department of Computer Science and Information Engineering
National Taiwan Normal University
Taipei, Taiwan 106, R.O.C
jlkoh@ntnu.edu.tw

Abstract. Recently, the data stream, which is an unbounded sequence of data elements generated at a rapid rate, provides a dynamic environment for collecting data sources. It is likely that the embedded knowledge in a data stream will change quickly as time goes by. Therefore, catching the recent trend of data is an important issue when mining frequent itemsets from data streams. Although the sliding window model proposed a good solution for this problem, the appearing information of the patterns within the sliding window has to be maintained completely in the traditional approach. In this paper, for estimating the approximate supports of patterns within the current sliding window, two data structures are proposed to maintain the average time stamps and frequency changing points of patterns, respectively. The experiment results show that our approach will reduce the run-time memory usage significantly. Moreover, the proposed FCP algorithm achieves high accuracy of mining results and guarantees no false dismissal occurring.

1 Introduction

The strategies for mining frequent itemsets in static databases have been widely studied over the last decade such as the Apriori[1], DHP[3], and FP-growth[2]. Recently, the data stream, which is an unbounded sequence of data elements generated at a rapid rate, provides a dynamic environment for collecting data sources. It is considered that the main restrictions of mining data streams include scanning the data in one pass and performing the mining within a limited memory usage.

Since it is not feasible to store the past data in data streams completely, a method for providing the approximate answers with accuracy guarantees is required. Lossy-counting is the representative approach for mining frequent itemsets from data streams[7]. Given an error tolerance parameter ε, the Lossy-counting algorithm prunes the patterns with support being less than ε from the pool of monitored patterns such that the required memory usage is reduced. Consequently, the frequency of a pattern is estimated by compensating the maximum number of times that the pattern

[*] This work was partially supported by the R.O.C. N.S.C. under Contract No. 94-2213-E-003-010 and 94-2524-S-003-014.

could have occurred before being inserted into the monitored patterns. It is proved no false dismissal occurs with Lossy-counting algorithm and the frequency error is guaranteed not to exceed a given error tolerance parameter. The hash-based approach was proposed in [4], in which each item in a data stream owns a respective list of counters in a hash table, and each counter may be shared by many items. A new novel algorithm, called hCount, was provided to maintain frequent items over a data stream and support both insertion and deletion of items with a less memory space.

Although the restriction of memory usage was considered in the two works introduced previously, the time sensitivity issue is another important issue when mining frequent itemsets from data streams. It is likely that the embedded knowledge in a data stream will change quickly as time goes by. In order to catch the recent trend of data, the *estDec* algorithm [5] decayed the old occurrences of itemsets as time goes by to diminish the effect of old transactions on the mining result of frequent itemsets in the data steam.

The above approach provided time-sensitive mining for long-term data. However, in certain applications, it is interested only the frequent patterns mined from the recently arriving data within a fixed time period. The sliding window method [6] attempted to solve this problem, in which the current sliding window was defined to be the most recently coming *W* transactions in a data stream by a given window size *W*. Accordingly, the recently frequent itemsets were defined to be the frequent itemsets mined from the current sliding window. Meanwhile, the Loosy-counting strategy was applied to reduce the number of maintained patterns. Consequently, the processing of the sliding window approach is characterized into two phases: window initialization phase and window sliding phase. The window initialization phase is activated when the sliding window is not full. In this phase, the occurrences of patterns in a newly coming transaction are maintained in a lattice structure at each time point. After the sliding window has become full, the window sliding phase is activated. In addition to maintain the occurrence for the new transaction, the oldest transaction has to be removed from the sliding window. However, all the transactions in the current sliding window need to be maintained in order to remove their effects on the current mining result when they are beyond the scope in the window.

To prevent from storing the whole transaction data in current transaction window, two data representation methods, named **average time stamps (ATS)** and **frequency changing points (FCP)**, respectively, are provided in this paper for monitoring the recent occurrence of itemsets in data streams. Moreover, a FP-tree-like data structure is adopted to store the monitored patterns for further reducing the memory usage. Accordingly, ATS and FCP algorithms are proposed for maintaining the corresponding monitoring data structures. The experimental results show that our approach will reduce the run-time memory usage significantly. Moreover, the proposed FCP algorithm achieves high accuracy of mining results and guarantees no false dismissal occurring.

This paper is organized as follows. The related terms used in this paper are defined in Section 2 first. The two provided methods for approximately monitoring recently frequent patterns in a data stream are introduced in Section 3 and Section 4, respectively. The performance evaluation on the proposed algorithms and a related work is reported in Section 5. Finally, in Section 6, we conclude this paper.

2 Preliminaries

Let $I = \{i_1, i_2, \ldots, i_m\}$ denote the set of items in the specific application domain and a transaction be composed of a bucket of items inputted within a fixed time unit. A data stream, $DS = [T_1, T_2, \ldots, T_t)$, is an infinite sequence of transactions, where each transaction T_i is associated with an time identifier i, and t denotes the time identifier of the latest transaction currently. Let the set of transactions in DS from time i to j be denoted as $DS[i,j]$. Under a predefined window size w, the ***current transaction window*** at time t, denoted as CTW_t, is $DS[t\text{-}w+1, t]$. The time identifier of the first transaction in CTW_t is denoted as CTW_t^{first}, that is $t\text{-}w+1$.

An itemset (or a pattern) is a set consisting of one or more items in I, that is, a non-empty subset of I. If itemset e is a subset of transaction T, we call T ***contains*** e. The number of transactions in CTW_t which contain e is named the ***recent support count*** of e in DS_t, denoted as $RC_t(e)$. The ***recent support*** of e, denotes as $Rsup_t(e)$, is obtained from $RC_t(e)$ / w. Given a user specified minimum support value between 0 and 1, denoted as S_{min}, and a maximum support error threshold value between 0 and S_{min}, denoted as ε, an itemset e is called a ***recently frequent*** itemset in DS_t if $Rsup_t(e) \geq S_{min}$. If $S_{min} \geq Rsup_t(e) \geq \varepsilon$, e is called a ***recently potential frequent*** itemset in DS_t. Otherwise, $Rsup_t(e) < \varepsilon$ and e is a ***recently infrequent*** itemset in DS_t.

3 Average Time Stamps Method

3.1 ATS Monitoring Data Structure

In our model of data streams, each transaction T_i, has a corresponding time identifier i. For an itemset p contained in T_i, i is named an ***appearing time stamp*** of p. The ***average time stamp*(*ATS*)** of an itemset p is the average of all the appearing time stamps of p from the first time p appears in the data stream.

In the ATS monitoring data structure, each entry maintains a 3-tuple (t_s, f, sum) for its corresponding itemset p as described as the following.

1) t_s: time of the first occurring of p to be counted into the accumulated support count;
2) f: the accumulated support count of p in $DS[t_s, t]$;
3) sum: the sum of appearing time stamps of p in $DS[t_s, t]$.

Consequently, the average time stamp of a pattern p, denoted as $avg_t(p)$, is obtained by performing $p.sum$ / $p.f$.

3.2 ATS Algorithm

In the window initialization phase, for each newly coming transaction T_t at time t, the work for maintaining T_t in the *ATS* monitoring data structure is described as follows. For each subset p of T_t, if the corresponding record is stored in the monitoring data structure, the record is updated to be: $p.f= p.f+1$ and $p.sum= p.sum+t$. Otherwise, a record for p is inserted into the data structure with $p.f =1$, $p.t_s= t$ and $p.sum= t$.

Similar to the window sliding phase proposed by [6], in addition to append the new transaction, the first transaction in $CTW_{t\text{-}1}$ has to be removed from the current transaction window. However, the transactions in CTW_{t-1} are not maintained in our approach. Instead, the average time stamps of itemsets are estimated according to the stored information. If $p.t_s < CTW_t^{first}$ and $avg_t(p) < p.t_s + (CTW_t^{first} - p.t_s)/2$, it implies most of the occurrences of p are beyond the period of CTW_t under the uniform distribution assumption. Therefore, such a pattern is pruned from the monitoring data structure. Otherwise, p is still active in CTW_t and the corresponding record remains.

Moreover, in order to reduce the memory requirement, the records of recently infrequent patterns are pruned from the monitoring data structure every m time identifiers. The recent support of a pattern p is estimated according to the following equation:

If $p.t_s > CTW_t^{first}$, $Rsup_t(p)=(p.f + \lfloor((p.t_s\ div\ m)\times m - CTW_t^{first}) \times\varepsilon\rfloor)/w$; ---- <1>
Else $Rsup_t(p)= p.f / (t- p.t_s +1)$. ---- <2>

Based on the information cashed in the monitoring data structure, the structure is traversed to find all the patterns p with $Rsup_t(p) \geq S_{min}$ whenever needing to mine the recently frequent itemsets.

4 Frequency Changing Points Method

4.1 FCP Monitoring Data Structure

With the average time stamp method described in Section 3, for each pattern p, $p.f$ is used to accumulate the occurrences of p from $p.t_s$ to current time t. If a pattern p was never being pruned from the monitoring data structure, $p.t_s$ denotes the first time identifier when p appeared in the data stream. There are two cases to be analyzed according to the relationship between $p.t_s$ and CTW_t^{first}:

(1) $p.t_s \geq CTW_t^{first}$: it is implied that $p.f$ accumulates the actual support count of p in CTW_t. Thus, $Rsup_t(p)$ is obtained from $(p.f / w)$ accurately.

(2) $p.t_s < CTW_t^{first}$: it is implied that $p.f$ accumulates the support count of p not only in CTW_t but also in $DS[p.t_s, CTW_t^{first}-1]$. The cases that whether a pattern p is frequent in $DS[p.t_s, CTW_t^{first}-1]$, in CTW_t, and in $DS[p.t_s, t]$ are summarized in Table 1.

Table 1. The case analysis of a frequent pattern in *DS* during different time intervals

Interval	$DS[p.t_s, CTW_t^{first}-1]$	CTW_t	$DS[p.t_s, t]$
Case 1	p is frequent	p is frequent	p is frequent
Case 2	p is frequent	p is infrequent	p is frequent or infrequent
Case 3	p is infrequent	p is frequent	p is frequent or infrequent
Case 4	p is infrequent	p is infrequent	p is infrequent

According to Case 2 and Case 3, to decide whether p is frequent in CTW_t according to its support in $DS[p.t_s, t]$ may cause false alarm and false dismissal, respectively. The false alarm occurring in Case 2 dues to p becomes sparser in CTW_t. The recent support of p in CTW_t is estimated according the count in $DS[p.t_s, t]$ such that p is evaluated to be recently frequent incorrectly. Similarly, in the situation of Case 3, the false dismissal occurring because a frequent pattern p in CTW_t is judged to be infrequent wrongly if p appears sparsely in $DS[p.t_s, CTW_t^{first}\text{-}1]$.

According to the cases discussed above, it is a critical point when the appearing frequency of a pattern becomes sparser. Let t' denote the last time identifier when a pattern p appeared previously. If p appears at current time t and $(t\text{-}t') > (1/S_{min})$, t is named a ***frequency changing point*(*FCP*)** of p. It means p is infrequent in $DS[t'\text{+}1, t]$. The frequency changing points of a pattern p are being used to adjust the boundaries of intervals for accumulating the support counts of p. When $p.t_s$ is beyond the corresponding time interval of CTW_t, $p.t_s$ is adjusted to be a frequency changing point of p as close to CTW_t^{first} as possible. Accordingly, the estimated recent support of p will approach the actual recent support of p in CTW_t.

In the monitoring data structure of frequency changing point, for an itemset p, each entry maintains a 5-tuple ($t_s, f, t_e, C_d, Rqueue$) as described as the following.

(1) t_s: the starting time of p to be counted into the accumulated support count;
(2) f: the accumulated support count of p in $DS[t_s, t]$;
(3) t_e: the time identifier of the most recent transaction that contains p;
(4) C_d: the accumulated support count of p in $DS[t_s, t_e\text{-}1]$;
(5) *Rqueue*: a queue consists of a sequence of (ct_1, ac_1), (ct_2, ac_2), …, (ct_n, ac_n) pairs, in which ct_i is a frequency changing point of p for i=1,…,n. Besides, ac_1 denotes the support count of p in $DS[t_s, ct_1\text{-}1]$ and ac_i denotes the support count of p in $DS[ct_{i-1}, ct_i\text{-}1]$ for i=2,…,n.

4.2 FCP Algorithm

In the window initialization phase, each subset p in a new transaction is appended into the FCP monitoring data structure (*FCP_MDS*). The maintained information of p includes its frequency changing points and the accumulated support counts between the changing points. The corresponding pseudo code is shown below.

```
Procedure AppendNew_FCP()
  {If $p$ is in FCP_MDS
     { $p.f = p.f+1$;
       If $t\text{-}p.t_e > (1/S_{min})$  /* a frequency changing point */
         {n = the number of elements in p.Rqueue ;  n=n+1;
         $ct_n=t$, $ac_n = (p.f - 1) - p.C_d$;
         append $(ct_n, ac_n)$ into p.Rqueue;  $p.C_d = p.f - 1$; }
     $p.t_e = t$;}
   else { $p.f=1$; $p.t_s= t$ ; $p.t_e= t$; $p.C_d=0$; p.Rqueue=null;
         insert p into FCP_MDS;}}
```

In the window sliding phase, in addition to perform procedure AppendNew_FCP(), it is necessary to adjust the starting point of support count accumulation for the monitored pattern p if $p.t_s$ is less than CTW_t^{first}. If *p.Rqueue* is empty for such a pattern p, it implies p remains frequent during the accumulation interval as Case 1 enumerated in Table 1. Therefore, no false dismissal occurs when discovering recently frequent patterns according to the support count of p in $DS[p.t_s, t]$ and there is no need to adjust $p.t_s$. On the other hand, it implies there is one or more frequency changing points of *p* occurring if *p.Rqueue* is not empty. Then the frequency changing points of *p* are checked one by one to adjust its starting time of support count accumulation. It is applicable to adjust $p.t_s$ in the following three situations:

(1) The changing point $ct_1 \le CTW_t^{first}$: it implies the support count accumulated in ac_1 is beyond the scope of CTW_t.

(2) The changing point $ct_1 > CTW_t^{first}$ and ac_1=1: it implies the occurring of *p* before ct_1 is at $p.t_s$ only which is out of the time period covered by CTW_t.

(3) The changing point $ct_1 > CTW_t^{first}$, ac_1>1, and the previous time point when *p* appears before ct_1, denoted as t_e', is less than CTW_t^{first}: although the value of t_e' is not maintained, the largest value of t_e' is derivable from ct_1 and $p.t_s$. Because ct_1 is a frequency changing point, $ct_1 - t_e' > 1/S_{min}$. Thus, $t_e' < (ct_1 - 1/S_{min})$. Moreover, ac_1 keeps the support count accumulated in $DS[p.t_s, ct_1\text{-}1]$ and no changing point occurs in this period. In other words, the interval of every two adjacent appearing time among the ac_1 times of occurring must be less than or equal to $1/S_{min}$. Thus, $t_e' \le p.t_s + (ac_1-1)\times(1/S_{min})$ and $t_e' < p.t_s + (ac_1-1)\times(1/S_{min})+1$ is derived. By combining these two inequalities, the largest value of t_e', denoted as t_e'*_max*, is derived to be $\min((ct_1 - 1/S_{min}), p.t_s + (ac_1-1)\times(1/S_{min})+1)-1$. If t_e'*_max* is less than CTW_t^{first}, it implies the support count accumulated in ac_1 has been expired.

When satisfying each one of the situations enumerated above, the changing point pair (ct_1, ac_1) is removed from *p.Rqueue*, $p.t_s$ is adjusted to be ct_1; and the accumulated support counts *p.f* and $p.C_d$ are modified accordingly. Then the following changing points are examined similarly. The corresponding code is shown below.

Procedure AdjustStart_FCP()

```
{For each p in FCP_MDS
    If $p.t_s < CTW_t^{first}$
    If Rqueue≠null {
      i=1;  Adjust = True;
      While (Adjust) {
              If ($ct_i \le CTW_t^{first}$)
                 Else if (($ct_i > CTW_t^{first}$)∧( $ac_i$=1))
                 Else {$t_e$'_max = min(($ct_1 - 1/S_{min}$), $p.t_s + (ac_1-1)\times(1/S_{min})+1) - 1$;
                       If ($t_e$'_max ≥ $CTW_t^{first}$) Adjust = False;}
              If (Adust) {
                 Remove ($ct_i$,$ac_i$) from p.Rqueue;
                 $p.t_s = ct_i$; p.f= p.f-$ac_i$; $p.C_d = p.C_d$-$ac_i$. ;   i= i +1; }
             }/*end while */
    } /* end If */ }
```

Moreover, it is indicated that a pattern *p* does not appear in CTW_t if *p.f* becomes 0 or t_e is less than CTW_t^{first}. Therefore, such a pattern is pruned to prevent from storing the unnecessary patterns in the monitoring data structure.

The situation that $p.t_s$ and *p.f* are not adjusted occurs when *p.Rqueue* is not empty and the changing point ct_1 in *p.Rqueue* does not satisfy the three situations enumerated above. It is implied that $ct_1 > CTW_t^{first}$, ac_1>1, and t_e'_max ≥ CTW_t^{first}. In this case, *p* is frequent in $DS[p.t_s, CTW_t^{first}\text{-}1]$ because there is not any frequency changing point appearing between $p.t_s$ and ct_1. When judging whether p is recently frequent in CTW_t according to its support estimated from $DS[p.t_s, t]$, even though false alarm may occur, it is certain that no false dismissal will occur.

Similar to the ATS algorithm, the recently infrequent patterns are pruned periodically to avoid the monitoring data structure glowing huge. The recent support of a pattern is estimated also according to equation <1> and <2> defined in ATS method.

[Example 1]

Table 2. Data stream sample

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Itemset	AB	AB	D	A	AB	AC	AE	AC	E	AE	AD	B	AE	B	AE

Table 2 shows a sample of data stream. Suppose S_{min} is set to be 0.5, ε is 0.25, and window size w is 10. Under the assumption that the infrequent patterns are pruned every 5 time identifiers, the process of constructing the monitoring data structure of frequency changing points is described as the following.

From time t_1 to t_4, there was not any frequency changing point occurring for those monitored patterns. The corresponding constructed monitoring data structure at t_4 is shown in Figure 1(a). After that, time t_5 is a frequency changing point of AB. Thus, the changing point entry (5, 2) is appended into AB.*Rqueue*. After pruning the infrequent pattern D, whose estimated recent support is less than 0.2, the resultant monitoring data structure is shown in Figure 1(b).Continuing the similar processing, the resultant monitoring data structure at t_{10} after the patterns in T_{10} have been appended into the data structure is shown in Figure 1(c). Then the following process starting at t_{11} changes into the window sliding phase. The values of $p.t_s$ and. $p.f$ are going to be adjusted if $p.t_s$ is less than CTW_t^{first}. For example, after the patterns in T_{12} are processed, both B.t_s and AB.t_s are less than CTW_{12}^{first}, and their corresponding *Rqueues* are not empty. By satisfying the third case among the three conditions of adjustment, B.t_s is adjusted to be 5 and B.f= B.f -2. Similarly, AB.t_s is set to be 5 and AB.f= AB.f-2. The obtained result is shown in Figure 1(d). After the patterns in T_{15} are appended to the monitoring data structure at time t_{15}, the information of pattern B is adjusted. Then pattern AB is removed because the last time it appeared is out of the range of CTW_{15}. After pruning the infrequent itemsets C, D, and AC, the monitoring data structure is shown as Figure 1(e).From the result shown in Figure 1(e), the recent supports of A, B, E, and AE are estimated to be 0.73(11/15=0.73), 0.3((2+1/10=0.3), 0.5(5/10=0.5), and 0.4(4/10 =0.4). Therefore, the discovered recent frequent patterns are A and E.

For compressing the FCP monitoring data structure in the implementation, the FP-tree-like structure is adopted to store the 5-tuples of patterns. However, to prevent from two scans over the data set, the items in a transaction are sorted according to their alphanumeric order instead of their frequency-descending order. Moreover, the mining algorithm on FP-tree is performed on *FCP_MDS* to find all the patterns p with $Rsup_t(p) \geq S_{min}$ on demand.

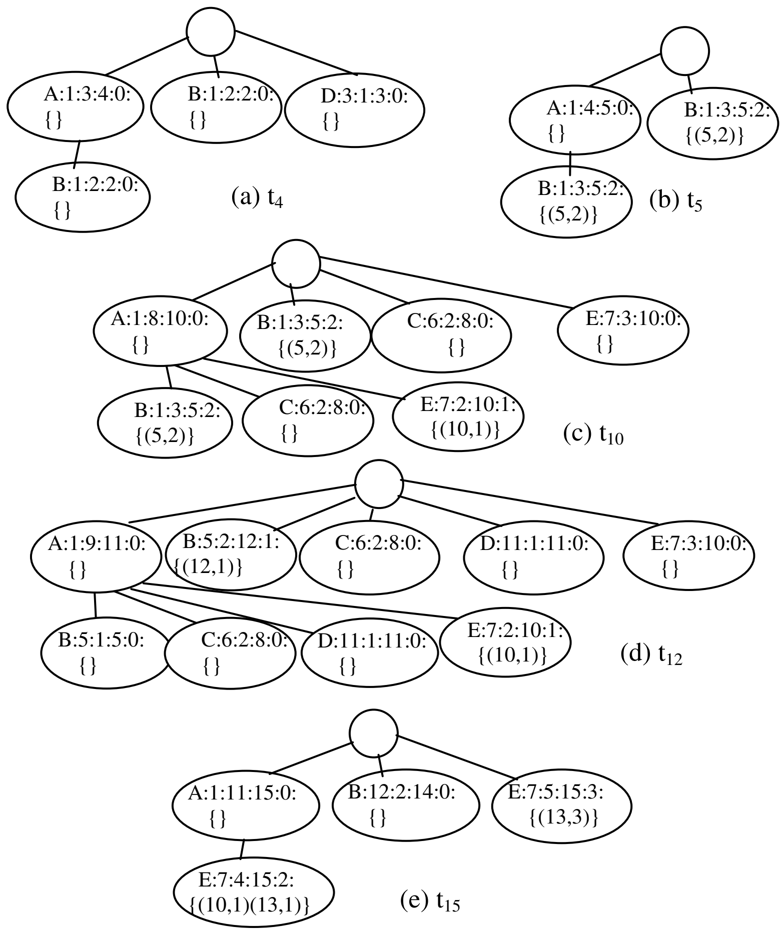


Fig. 1. The *FCP_MDS* of the data stream sample

5 Performance Study

The proposed algorithms and Sliding Window method (SW algorithm in short) [6] are implemented using Visual C++ 6.0. The TD_FP_Growth algorithm[8] is applied to discover recently frequent patterns from the FCP monitoring data structure. The experiments have been performed on a 3.4GHz Intel Pentium IV machine with 512 megabytes main memory and running Microsoft XP Professional. Moreover, the data sets are generated from the IBM data generator [1], where each dataset simulates a data stream with a transaction coming within each time unit. In the first part of experiments, the false dismissal rates/ false alarm rates are measured to indicate the effectiveness of the proposed methods. Furthermore, the execution time and memory usage is measured in the second part of experiments to show the efficiency of the proposed FCP algorithm by comparing with the ones of SW algorithm.

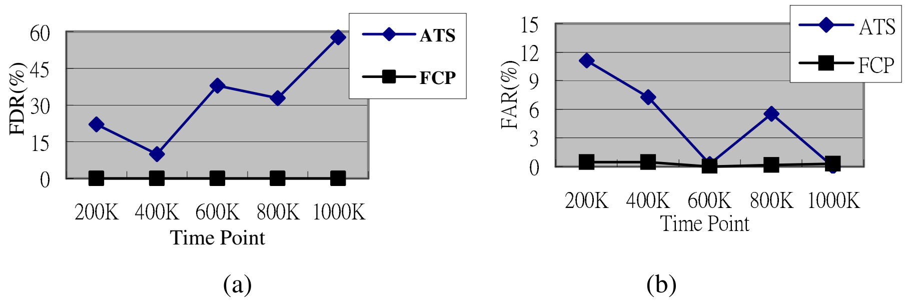


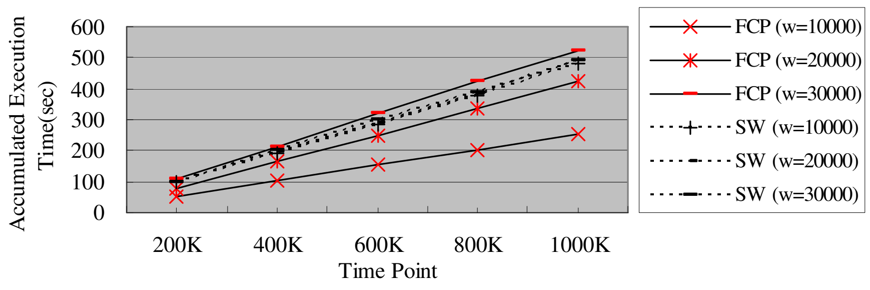
Fig. 2. The FDR and FAR values of the mining results

[Experiment 1]. To evaluate the effectiveness of ATS and FCP algorithms, an experiment is performed on the dataset T5.I4.D1000K with window size=20000, S_{min} =0.01, and ε =0.005. In this experiment, ATS and FCP algorithms are performed to maintain the corresponding monitoring data structures, respectively, and TD_FP-Growth algorithm is performed once every 200K time points to find recently frequent itemsets. By comparing the mining results with the frequent itemsets found by Apriori Algorithm on the corresponding CTW_t, the false dismissal rate (FDR), false alarm rate (FAR), and average support error (ASE) of the two algorithms are measured.

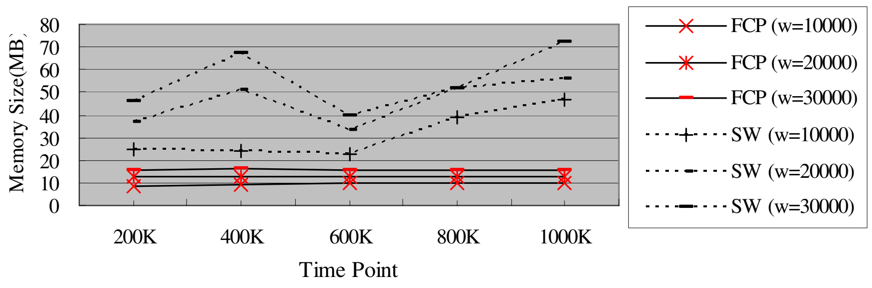
The results shown in Figure 2(a) illustrates that all the recently frequent patterns are discovered and no false dismissal occurs in FCP algorithm. On the other hand, the false dismissal rate of ATS algorithm changes dramatically, it is indicated that its mining quality is unstable. The false alarm rates of the two proposed algorithm at various time points are shown in Figure 2(b). It is reported that the false alarm rate of FCP algorithm is below 0.5%. Although the ATS algorithm has higher FAR, more than 88% of the mining results are accurate.

Moreover, the ***average support error*** defined in [7] is also used to model the relative accuracy of the proposed methods. The results show that ASEs($R_{\text{FCP}}|R_{\text{Apriori}}$) keeps under 3×10^{-5} at different time points; ASEs($R_{\text{FCP}}|R_{\text{Apriori}}$) are less than 3×10^{-3} but with variations from time to time.

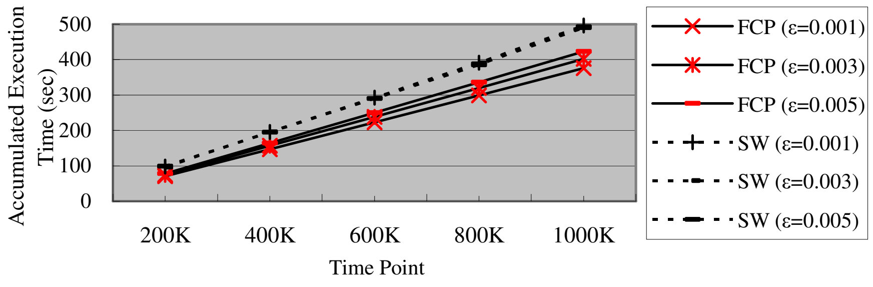
[Experiment 2]. In this experiment, FCP and SW algorithms are compared on their accumulated execution time and maximum memory usage used for maintaining the monitored patterns (the mining time is not included). This experiment is performed on the dataset T5.I4D1000K with S_{min}=0.01 and ε =0.005. When the window size is varied from 10000, 20000, to 30000, the results of accumulated execution time and maximum memory usage are shown in Figure 3(a) and 3(b), respectively. Although the accumulated execution time increases as the window size is raised, the efficiency of FCP algorithm is comparable to the one of SW algorithm. In contrast with SW algorithm, the memory usage of FCP is significantly reduced without being sensitive to the window size. Moreover, it verifies that FCP is feasible for the streaming environment with a small memory. By varying the setting of ε, Figure 3(c) and 3(d) show the results of accumulated execution time and maximum memory usage, respectively. It is indicated that the execution time of FCP is not sensitive to ε. In addition, the maximum memory usage of FCP keeps steady, which is limited within 15 MB.



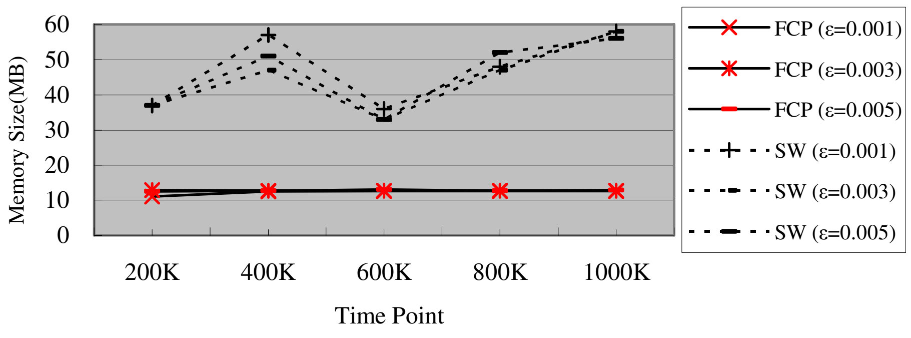
(a)



(b)



(c)



(d)

Fig. 3. The execution time and memory usage of FCP and SW algorithms

6 Conclusion

In this paper, the average time stamps and frequency changing points are proposed, respectively, to represent the summarization of occurrences of recent patterns. Consequently, ATS and FCP algorithms are designed for maintaining the corresponding FP-tree-like monitoring data structures according to the newly coming transaction. Besides, the effect of old transactions on the mining result of recent frequent itemsets is diminished by performing pruning rules on the monitoring data structures without needing to keep the whole transactions in the current sliding window physically. From the monitoring data structure, the recently frequent patterns are discovered efficiently at any time. Finally, the experimental results demonstrate that the proposed FCP algorithm achieves high accuracy for approximating the supports of recently frequent patterns and guarantees no false dismissal occurring. Not only the execution time of FCP is acceptable under various parameters setting, but also the memory usage of FCP is significantly reduced by comparing with the one of SW algorithm. It demonstrates that FCP is a stable and feasible algorithm for mining recently frequent patterns in the streaming environment with a small memory.

References

1. R. Agrawal and R. Srikant, “Fast Algorithms for Mining Association Rules,” in Proc. of Int. Conf. on Very Large Data Bases, 1994.
2. J. Han, J. Pei, Y. Yin and R. Mao, “Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach”, Data Mining and Knowledge Discovery, 8(1):53-87, 2004.
3. J.S. Park, M.S. Chen, and P.S. Yu, “An Effective Hash-based Algorithm for Mining Association Rules,” in Proc. of the ACM SIGMOD International Conference on Management of Data (SIGMOD'95), May, pages 175-186, 1995.
4. C. Jin, W. Qian, C. Sha, J. X. Yu, and A. Zhou, “Dynamically Maintaining Frequent Items Over a Data Stream,” in Proc. of the 12th ACM International Conference on Information and Knowledge Management, 2003.
5. J. H, Chang and W.S. Lee, “Finding Recent Frequent Itemsets Adaptively over Online Data Streams, ” in Proc. of the 9th ACM International Conference on Knowledge Discovery and Data Ming, 2003.
6. 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, pp753-762, 2004.
7. G. S. Manku and R. Chen Motwani, “Approximate Frequent Counts over Data Streams, ” in Proc. of the 28th International Conference on Very Large Database, Hong Kong, China Aug, 2002.
8. K Wang, L. Tang, J. Han, and J. Liu, “Top Down FP-Growth for Association Rule Mining, ” in Proc. of the 6th Pacific Area Conference on Knowledge Discovery and Data Mining, May 6-8, Taipei, Taiwan, PAKDD-2002.