A Simple but Effective Maximal Frequent Itemset Mining Algorithm over Streams

Haifeng Li, Ning Zhang, Zhixin Chen
School of Information, Central University of Finance and Economics, Beijing China, 100081
Email: mydlhf@126.com, zhangning75@gmail.com, czx.bupt@gmail.com

***Abstract*—Maximal frequent itemsets are one of several condensed representations of frequent itemsets, which store most of the information contained in frequent itemsets using less space, thus being more suitable for stream mining. This paper considers a simple but effective algorithm for mining maximal frequent itemsets over a stream landmark. We design a compact data structure named *FP-FOREST* to improve an state-of-the-art algorithm *INSTANT*; thus, itemsets can be compressed and the support counting can be effective performed. Our experimental results show our algorithm achieves a better performance in memory cost and running time cost.**

I. INTRODUCTION

Frequent itemset mining is a traditional and important problem in data mining. An itemset is frequent if its support is not less than a threshold specified by users. Traditional frequent itemset mining approaches have mainly considered the problem of mining static transaction databases. In these methods, transactions are stored in secondary storage so that multiple scans over the data can be performed. Three kinds of frequent itemset mining approaches over static databases have been proposed: reading-based[3], writing-based[15], and pointer-based[18]. [14] presented a comprehensive survey of frequent itemset mining and discussed research directions.

Many methods focusing on frequent itemset mining over a stream have been proposed. [13] proposed *FP-Stream* to mine frequent itemsets, which was efficient when the average transaction length was small; [22] used lossy counting to mine frequent itemsets; [7],[8], and [9] focused on mining the recent itemsets, which used a regression parameter to adjust and reflect the importance of recent transactions; [27] presented the *FTP-DS* method to compress each frequent itemset; [10] and [1] separately focused on multiple-level frequent itemset mining and semi-structure stream mining; [12] proposed a group testing technique, and [17] proposed a hash technique to improve frequent itemset mining; [16] proposed an in-core mining algorithm to speed up the runtime when distinct items are huge or minimum support is low; [19] presented two methods separately based on the average time stamps and frequency-changing points of patterns to estimate the approximate supports of frequent itemsets; [5] focused on mining a stream over a flexible sliding window; [20] was a block-based stream mining algorithm with *DSTree* structure; [23] used a verification technique to mine frequent itemsets over a stream when the sliding window is large; [11] reviewed the main techniques of frequent itemset mining algorithms over data streams and classified them into categories to be separately addressed. Given these algorithms, the runtime could be reduced, but the mining results were huge when the minimum support was low; consequently, the condensed representations of frequent itemsets including closed itemsets[25], maximal itemsets[31], free itemsets[4], approximate k-sets[2], weighted itemsets[28], and non-derivable itemsets[6] were proposed; in addition, [26] focused on discovering a minimal set of unexpected itemsets.

The concept of maximal frequent itemsets(*MFI*) was first proposed in 1998, an itemset is maximal frequent itemset if its support is frequent and it is not covered by other frequent itemsets, we will discuss the details in Section 2. Maximal frequent itemsets are one of the condensed representations, which only store the non-redundant cover of frequent itemsets, resulting in space cost reduction.

Many maximal frequent itemset mining algorithms were proposed to improve the performance. The main considerations focused on developing new data retrieving methods, new data pruning strategies and new data structures. Yang used directed graphs in [35] to obtain maximal frequent itemsets and proved that maximal frequent itemset mining is a $\sharp$p problem. The basic maximal frequent itemset mining method is based on the *a priori* property of the itemset. The implementations were separated into two types: One type is an improvement of the *a priori* mining method, a bread first search[32], with utilizing data pruning, nevertheless, the candidate results are huge when an itemset is large; a further optimization was the down-top method, which counted the weight from the largest itemset to avoid superset checking, also, the efficiency was low when the threshold was small. Another one used depth first search[33] to prune most of the redundant candidate results, which, generally, is better than the first type. In these algorithms, many optimized strategies were proposed[34][31]: The candidate group built a head itemset and a tail itemset, which can quickly built different candidate itemsets; the super-itemset pruning could immediately locate the right frequent itemset; the global itemset pruning deleted all

This research is supported by the National Science Foundation of China(61100112), and Program for Innovation Research in Central University of Finance and Economics.

the sub-itemsets according to the sorted itemsets; the item dynamic sort strategy built heuristic rules to directly obtain the itemsets with high support, which was extended by a further pruning based on tail itemset; the local check strategy got the related maximal frequent itemsets with the current itemset.

Recently, many stream mining algorithms for maximal frequent itemsets were proposed. [21] proposed an incremental algorithm *estDec+* based on *CP-tree* structure, which compressed several itemsets into one node according to their frequencies; thus, the memory cost can be flexibly handled by merging or splitting nodes. Furthermore, employing an *isFI-forest* data structure to maintain itemsets, [30] presented *DSM-MFI* algorithm to mine maximal frequent itemsets. Moreover, considering maximal frequent itemset is one of the condensed representation, [24] proposed *INSTANT* algorithm, which stored itemsets with frequencies under a specified absolute minimum support, and compared them to the new transactions to obtain new itemsets; Plus, [29] presented an improved method *estMax*, which predicted the type of itemsets with their defined maximal life circle, resulting in advanced pruning.

In this paper, we explore a new stream maximal frequent itemsets mining algorithm named *INSTANT+* based on a stream landmark model. The main contributions are as follows:

1) First, we design a compact and simple data structure named *FP-FOREST* to store the maximal frequent itemsets and infrequent itemsets. In *FP-FOREST*, each tree dynamically maintains the itemsets with equal support; thus, we can compress the storage cost, and efficiently compute the support.
2) Second, when new transaction arrives, we present a simple but efficient algorithm to dynamically maintain the *FP-FOREST*, in which itemsets are pruned effectively and the maximal frequent itemsets can be obtained in real time.
3) Finally, we evaluate the *INSTANT+* algorithm on two datasets in comparison to the state-of-the-art maximal frequent itemset mining method *INSTANT*. The experimental results show that *INSTANT+* is effective and efficient.

The rest of this paper is organized as follows: In *Section 2* we present the preliminaries of frequent itemsets and maximal frequent itemsets and define the mining problem. *Section 3* illustrates our data structure and our algorithm. *Section 4* evaluates the performance of *INSTANT+* with experimental results. Finally, *Section 5* concludes this paper.

II. Preliminaries and Problem Statement

A brief review of frequent itemset and maximal frequent itemset is presented in this section; based on the concepts, we introduce the problem addressed in this paper.

TABLE I
SIMPLE DATABASE

id	itemsets
1	b c d e
2	a b c d e
3	a b
4	a b d e
5	a c e

A. Preliminaries

1) Frequent Itemsets: Given a set of distinct items $\Gamma = \{i_1, i_2, \cdots, i_n\}$ where $|\Gamma| = n$ denotes the size of Γ, a subset $X \subseteq \Gamma$ is called an itemset; suppose $|X| = k$, we call X a k-itemset. A concise expression of itemset $X = \{x_1, x_2, \cdots, x_m\}$ is $x_1x_2\cdots x_m$. A database $D = \{T_1, T_2, \cdots, T_v\}$ is a collection wherein each transaction is a subset of Γ, namely an itemset. Each transaction $T_i (i = 1 \cdots v)$ is related to an id, i.e., the id of T_i is *i*. The absolute support (*AS*) of an itemset α, also called the weight of α , is the number of transactions which cover α, denoted $\Lambda(\alpha) = \{|T||T \in D \wedge \alpha \subseteq T\}$; the relative support (*RS*) of an itemset α is the ratio of *AS* with respect to $|D|$, denoted $\Lambda_r(\alpha) = \frac{\Lambda(\alpha)}{|D|}$. Given a absolute minimum support λ, itemset α is frequent if $\Lambda(\alpha) \geq \lambda$.

2) Maximal Frequent Itemsets: A maximal itemset is a largest itemset in a database D, that is, it is not covered by other itemsets. A maximal frequent itemset is both maximal and frequent in D.

Definition 1. Given the minimum support λ, an itemset α is an maximal frequent itemset if it is frequent and it is not covered by other frequent itemsets, denoted $\Lambda(\alpha) \geq \lambda \wedge \nexists\beta|(\beta \supset \alpha \wedge \Lambda(\beta) \geq \lambda)$.

Example 1. Given a simple database D as shown in Tab.I and the minimum support 2, the frequent itemsets are {a,b,c,d,e,ab,ac,ad,ae,bc,bd,be,cd,ce,de,ace,bcd,bce,bde,cde, abde,bcde}, nevertheless, the maximal frequent itemsets are {ace,abde,bcde}. As can be seen, the maximal frequent itemsets are much less than the frequent itemsets.

B. Problem Definition

We choose the landmark model in this paper because it can reflect the global characteristic of a stream, i.e., the problem addressed in this paper is to generate maximal frequent itemsets from the first arrived transaction to the most recently arrived one. Fig. 1 is an example with $\Gamma = \{a, b, c, d, e\}$. The initial window includes 1 itemsets, when transactions arrive, new itemsets are added.

III. *INSTANT+*

In this section, we will first describe the algorithm we need to improve, with presenting its drawbacks, we introduce an efficient data structure to address the problem and propose our solution with theoretical analysis.

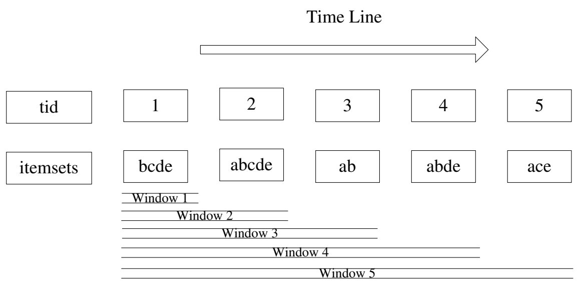


Fig. 1. A running example of landmark model

A. INSTANT *Introduction*

INSTANT is a state-of-the-art algorithm for mining maximal frequent itemsets over stream landmark model, which used a very simple but effective data structure to maintain the maximal frequent itemsets and infrequent itemsets.

INSTANT employs arrays U to store different itemsets with the equal support, that is, $U(i)$ is used to store the itemsets with support i; thus, given the minimum support λ, arrays from $U(1)$ to $U(\lambda)$ store the itemsets with support from 1 to λ, in which $U(\lambda)$ store the maximal frequent itemsets. When a new transactions T arrives, *INSTANT* will compare T to each itemset I in $U(i-1)$ and insert the interaction $T \cap I$ into $U(i)$ if it is not one of the itemsets in $U(i)$.

As an example, giving the minimum support is $\lambda = 3$, i.e., *INSTANT* maintains 3 arrays: $U(1)$, $U(2)$ and $U(3)$. Suppose the new arriving transaction T is abd, and $U(3) = \{ac\}$, $U(2) = \{abc, acd\}$ and $U(1) = \{abcd\}$, the procedure is as follows.

1) First, T compares with each itemset in $U(2)$, which is $\{abc, acd\}$, and obtains the interactions {ab,ad}, and insert them into $U(3)$ since ac do not cover ab and ad, and ab and ad do not cover ac. Thus, $U(3) = \{ac, ab, ad\}$.
2) Second, T compares with each itemset in $U(1)$, which is $\{abcd\}$, and obtains the interactions {abd}, and insert them into $U(2)$ since abd do not cover abc and acd, and abc and acd do not cover abd. Thus, $U(3) = \{abc, acd, abd\}$.
3) Finally, the itemsets $\{ac, ab, ad\}$ in $U(3)$ is output as the maximal frequent itemsets.

INSTANT is an efficient and effective algorithm, but we argue that it has drawbacks in two aspects. On the one hand, even though it only stores the useful itemsets, but the storage is not optimized, which results in a storage redundant; in the previous example, when $U(3)$ need to store $\{abc, acd, abd\}$, the a was stored repeatedly. On the other hand, the itemset comparison is a very regular operation, whose computation cost has a direct effect on the final running time cost, but it is also not optimized in *INSTANT*.

B. FP-FOREST

In our algorithm, computing the absolute support of each itemset is an usual action, we employ a *FP-FOREST*, which is actually composed of FP-trees, to store all the transactions of stream, which can reduce the memory cost and speedup the support computation. FP-tree[15] is a tree storing transactions in a root-path manner, that is, the common prefix of transactions can be stored once, and its repeated time can be recorded by a counter. As can be seen from the left image of Fig.2, FP-tree is a compact data structure, which can reduce the memory cost a lot.

We improved the FP-tree structure for our stream mining; thus, it has its own characteristics.

First, since each FP-tree stores the itemsets with equal support, in which the nodes have no item counters; as a result, our FP-tree can be incrementally maintained, i.e., when a new transaction arrives, either new branches are added, or existing nodes are pruned.

Second, our FP-tree node has no item counter, consequently, each branch is not covered by others; thus, once a new added itemset I covered by any branch, it will be pruned no matter where I locates. For example, as shown in the left image of Fig.2, in traditional FP-tree, ab is an independent branch, which is covered by $abcde$, hence, it is not presented in our first FP-tree.

Third, to recognize the transactions tid, our implementation is based on an appended tid-set, which stores the transactions tid, and each tid points the right position of FP-tree. We can see from the right image of Fig.2 that the bottom numbers are the tids. Since our FP-tree only stores the maximal branches, the tids distribute at different FP-trees. For example, we can find a transaction with tid=2 in our first FP-tree in the right image of Fig.2; thus, we can retrieve from node a(which is pointed by tid 2) to the root, and get the corresponding transaction $abcde$.

C. INSTANT+ *Algorithm*

Given the absolute minimum support λ, our algorithm improves *INSTANT* based on the proposed data structure *FP-FOREST*, which is implemented as a series of *FP-trees*, denoted $FP_i (1 \leq i \leq \lambda)$.

1) INSTANT+ *Implementation and Complexity Analysis:* Employing *FP-FOREST*, our algorithm is a little different from *INSTANT* when we compare and update the itemsets, resulting in a great efficiency improvement.

1) *INSTANT* need to compare the new arrived transaction T to all existing itemsets, that is, given the itemsets(the number is n) with equal support, *INSTANT* will perform n comparisons, i.e., the time complexity is $O(n)$; if the average size of these n itemsets is m, then the average time complexity is $O(m)$; thus, the overall time complexity is $O(mn)$. Nevertheless, in *INSTANT+*, the itemsets with equal support have been compressed in a *FP-tree* and indexed, if an itemset has no interaction with T, the computation is ignored; suppose the the number of related itemsets is $w(w << n)$, then the time complexity is also $O(mw)$.

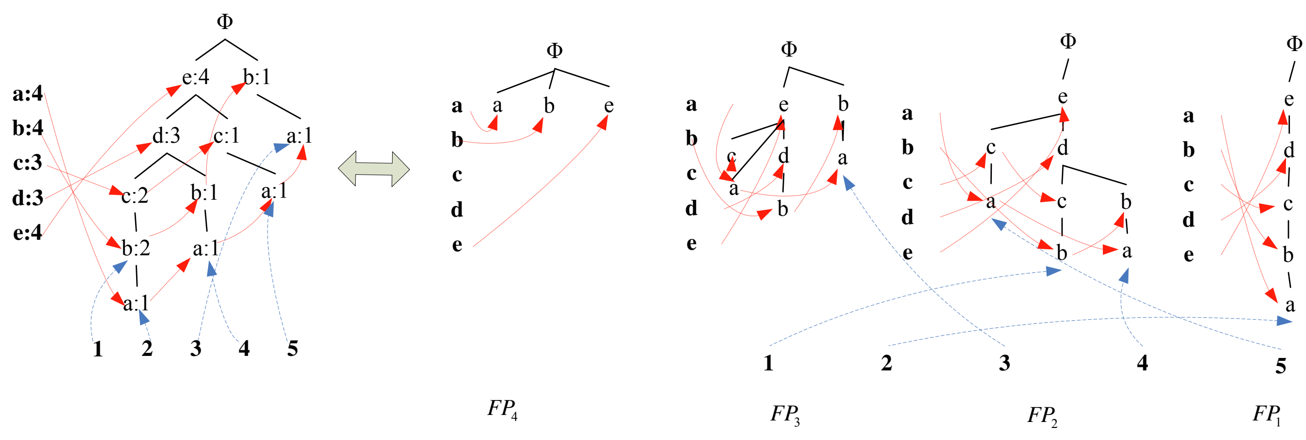


Fig. 2. Traditional FP-Tree V.S. Our FP-Tree

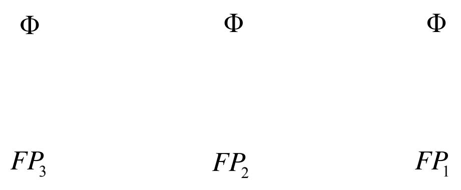


Fig. 3. Initial FP-FOREST

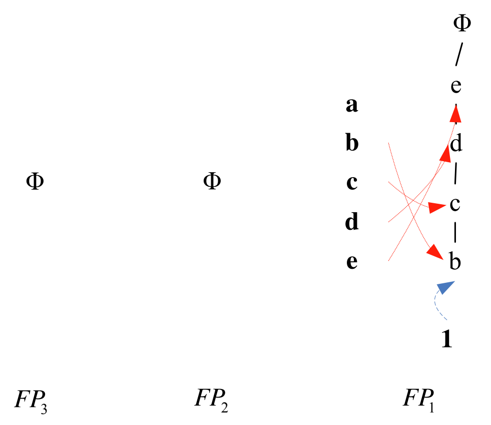


Fig. 4. FP-FOREST after adding *bcde*

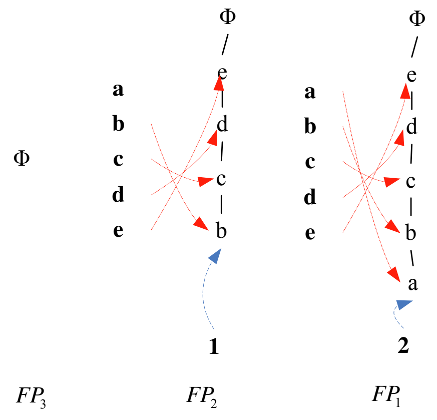


Fig. 5. FP-FOREST after adding *abcde*

2) *INSTANT* need to add and delete the itemset, which are based on the itemset comparison, that is, when adding an itemset, *INSTANT* will compare it with all the existing itemsets and insert it if it is not exist, so is deleting an itemset; thus, even though their own time complexity is $O(1)$, the overall time complexity is $O(mn)$. On the other hand, when adding or deleting an itemset, we will traverse *FP-tree* twice, the first time is to deciding the itemset existence, the second time will add or delete the itemset; thus, the overall time complexities are $O(mw)$.
3) *INSTANT* stores all the individual itemsets, that is, given the itemsets(the number is n) with equal support, the space complexity is $O(n)$. Our algorithm uses *FP-FOREST*, which can share the items at the same branch, the space complexity decreases to $O(logn)$.

2) An Example: We use an example to describe the mining process in detail. The dataset in Fig.1 is employed in our example. As can be seen, the transactions are added one by one, and we will dynamically maintain the $FP-FOREST$.

Suppose the absolute minimum support $\lambda = 3$, i.e., 3 $FP-trees$, FP_1, FP_2 and FP_3, are in our structure. The mining process is as follows.

1) First, as shown in Fig.3, when no transaction arrives, all $FP-trees$ are empty, denoted Φ.
2) Second, as shown in Fig.4, the first transaction *bcde* arrives, then FP_1 is updated, whereas FP_2 and FP_3 are still null.
3) Third, as shown int Fig.5, the second transaction *abcde* arrives, it compares to *bcde* in FP_1, and inserts the interaction *bcde* into FP_2; then, *bcde* is pruned from FP_1 and *abcde* is inserted into FP_1.
4) Fourth, as shown int Fig.6, the third transaction *ab*

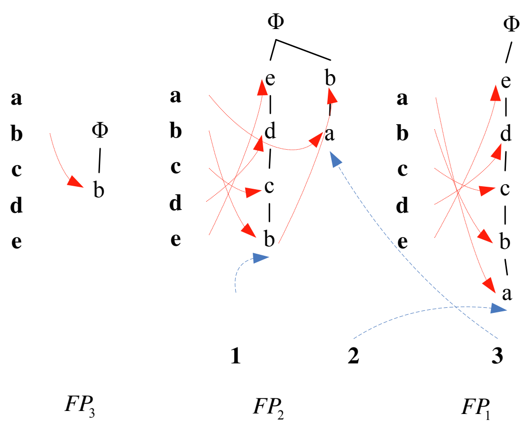


Fig. 6. FP-FOREST after adding *ab*

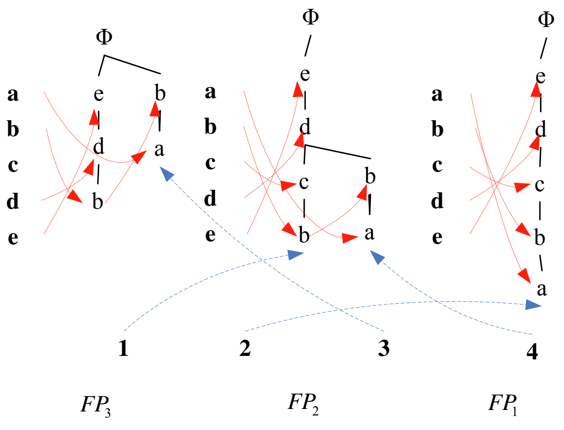


Fig. 7. FP-FOREST after adding *abde*

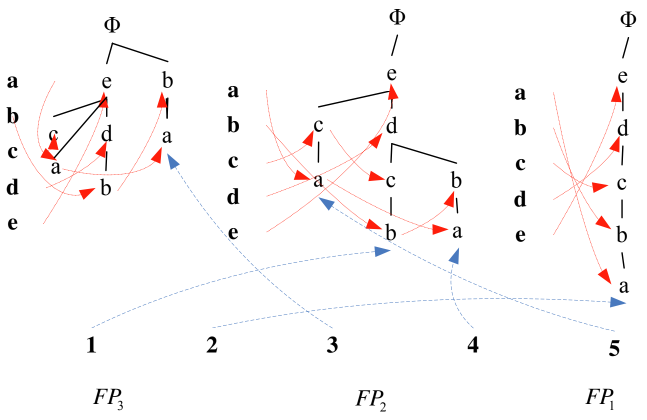


Fig. 8. FP-FOREST after adding *ace*

TABLE II
DATA CHARACTERISTICS

DataSet	nr. of trans.	avg. trans. length	min. trans. length	max. trans. length	nr. of items
T40I10D100K	100 000	39.6	4	77	942
MUSHROOM	8 124	23	23	23	119

arrives, it compares to $bcde$ in FP_2, and inserts the interaction b into FP_3; then, ab compares to $abcde$ in FP_1 and inserts the interaction ab into FP_2.

5) Fifth, as shown int Fig.7, the fourth transaction $abde$ arrives, it compares to $bcde$ and ab in FP_2, and inserts the interaction bde and ab into FP_3 with b being pruned from FP_3; then, $abde$ compares to $abcde$ in FP_1 and inserts the interaction $abde$ into FP_2 with ab being pruned from FP_2.
6) Finally, as shown int Fig.8, the fifth transaction ace arrives, it compares to $bcde$ and $abde$ in FP_2, and inserts the interaction ce and ae into FP_3; then, ace compares to $abcde$ in FP_1 and inserts the interaction ace into FP_2.

As can be seen, the final maximal frequent itemsets are in FP_3, which are ae,ce,ab and bde. The items count decreases from 25 in *INSTANT* to 20 in *INSTANT+*.

IV. Experimental Results

We conducted a series of experiments to evaluate the performance of *INSTANT+*. The state-of-the-art mining algorithm, *INSTANT*[24], was used as the evaluation method.

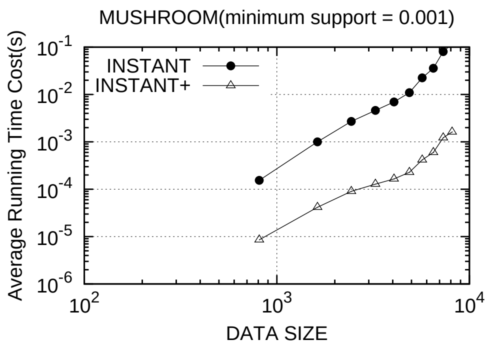
A. Running Environment and Datasets

All experiments were implemented with $C\sharp$, compiled with *Visual Studio 2005* in *Windows Server 2003* and executed on a *Xeon 2.0GHz* PC with *2GB* RAM.

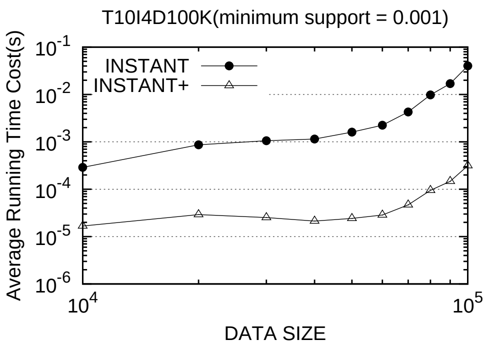
We used 1 synthetic datasets and 1 real-life datasets, which are well-known benchmarks for frequent itemset mining. The *T40I10D100K* dataset is generated with the IBM synthetic data generator. The *MUSHROOM* dataset contains characteristics from different species of mushrooms. The data characteristics are summarized in *Tab. 2*.

B. Running Time Cost Evaluation

We firstly compared the average runtime of these two algorithms under different data sizes when the minimum support was fixed. As shown in all of the images in Fig.9, the running time cost of *INSTANT* and *INSTANT+* increase following the data size; these results verify that both algorithms are sensitive to data size, which is due to the using of unchanged absolute minimum support: When more transactions arrive, more itemsets are generated and compared to the new itemset, since *INSTANT* has no optimized method, the running time cost increase hugely; on the contrary, *INSTANT* uses the *FP-FOREST* as the index, resulting in the whole running time cost reduction,



(a) Runtime cost vs. number of records



(b) Runtime cost vs. number of records

Fig. 9. Runtime Cost

and even though the running time cost increases following the data size, the increasing scope is small. Also, from the two figures we can see that the average transaction size has an significant effect on the running time cost, since itemset comparisons and interactions are performed regularly: As can be seen, the running of both algorithms over *MUSHROOM* dataset are slower than running over *T40I10D100K* dataset, that is, the larger the average transaction size, the worse the running time cost.

We also compared the average runtime of these two algorithms under different minimum supports when data sizes was fixed. As shown in all of the images Fig.10, the running time cost of *INSTANT* and *INSTANT+* are almost unchanged when the minimum support become large; but in comparison, the runtime cost of *INSTANT+* is much lower than that of *INSTANT*. In detail, the running time of *INSTANT* always raises, that is because when the minimum support turns large, *INSTANT* need to store more infrequent itemsets; thus, the itemset search and comparison cost increases. Whereas the runtime cost of *INSTANT+* over *MUSHROOM* dataset reduces following the minimum support, which is due to the dataset characteristics: When new maximal frequent itemset are generated, some itemsets will be pruned and the memory usage is further compacted, resulting in the computation reduction.

C. Memory Cost Evaluation

To evaluate the memory cost of our algorithm, we compared the count of generated items in *INSTANT* and *INSTANT+*. As shown in Fig.11, when we fix the minimum support and increase the data size, the generated items of both algorithm become more, but the overall items count of *INSTANT+* is less than that of *INSTANT* since *INSTANT+* uses the *FP-FOREST* to prune the redundant items. Furthermore, when the minimum support increases, the incremental scope of *INSTANT+* is much reduced than that of *INSTANT*, that is because *FP-FOREST* will prune more redundant items when the overall items count is fixed but the itemsets count increases.

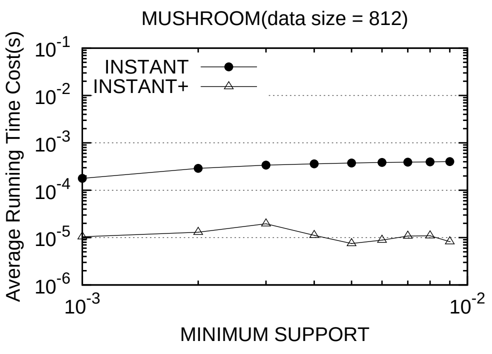
We also compared the items count of these two algorithms under different minimum supports when data size was fixed. As show in Fig.12, corresponding to the runtime cost in Fig.10, the items count in *INSTANT* is in direct proportion to the minimum support, but in *INSTANT+*, the items count may decreases, which is analyzed in the running time cost comparison.

V. Conclusions

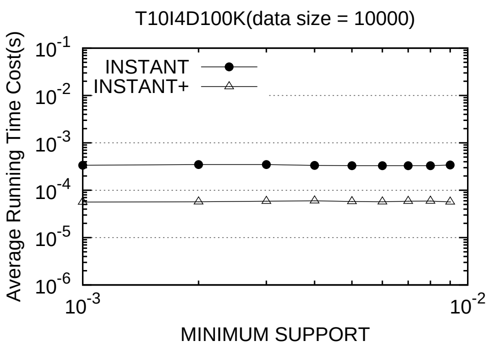
In this paper we considered a problem that how to mine maximal frequent itemset over stream landmark model. We discussed the drawbacks of an state-of-the-art algorithm *INSTANT* and introduced a compact data structure *FP-FOREST*, which can compress the itemset storage and optimize itemset computation, based on which, an improved algorithm *INSTANT* was developed and analyzed. Our experimental studies showed that our algorithm is effective and efficient.

References

[1] T.Asai, H.Arimura, K.Abe, S.Kawasoe, and S.Arikawa, Online Algorithms for Mining Semi-structured Data Stream, in: Proc. ICDM'2002
[2] F.Afrati, A.Gionis, and H.Mannila, Approximating a Collection of Frequent Sets, in: Proc. SIGKDD'2004
[3] R.Agrawal, and R.Srikant, Fast algorithms for mining association rules, in: Proc. VLDB'1994.
[4] J.Boulicaut, A.Bykowski, and C.Rigotti, Free-sets: a condensed representation of boolean data for the approximation of frequency queries, Data Mining and Knowledge Discovery, 7 (2003) 5-22
[5] T.Calders, N.Dexters, and B.Goethals, Mining Frequent Itemsets in a Stream, in: Proc. ICDM'2007
[6] T.Calders, and B.Goethals, Mining All Non-Derivable Frequent Itemsets, in: Proc. PKDD'2002
[7] J.H.Chang, and W.S.Lee, Decaying Obsolete Information in Finding Recent Frequent Itemsets over Data Stream, IEICE Transaction on Information and Systems, 87 (6) (2004) 1588-1592
[8] J.H.Chang, and W.S.Lee, A Sliding Window Method for Finding Recently Frequent Itemsets over Online Data Streams, Journal of Information Science and Engineering, 20 (4) (2004) 753-762

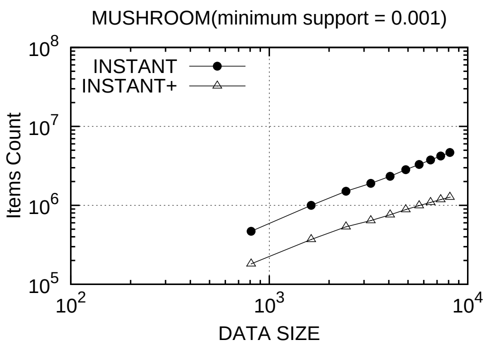


(a) Runtime cost vs. minimum support

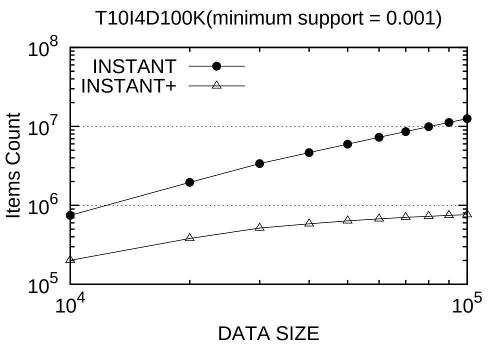


(b) Runtime cost vs. minimum support

Fig. 10. Runtime Cost

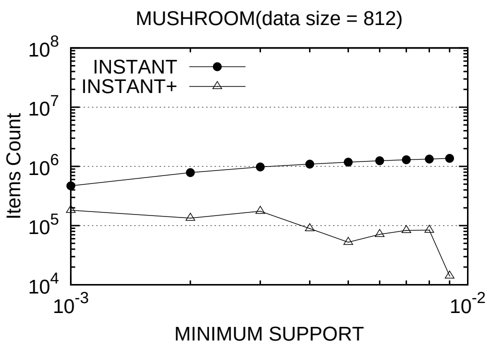


(a) Itemsets Count vs. number of records

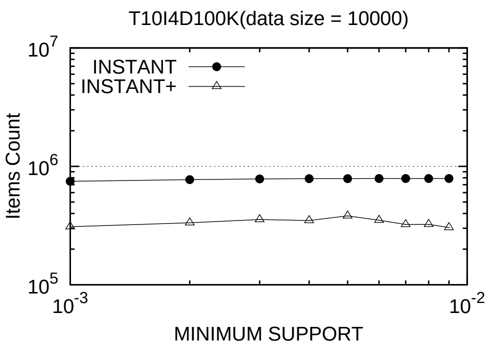


(b) Itemsets Count cost vs. number of records

Fig. 11. Memory Cost



(a) Itemsets Count vs. minimum support



(b) Itemsets Count cost vs. minimum support

Fig. 12. Memory Cost

[9] J.H.Chang, and W.S.Lee, Finding Recent Frequent Itemsets Adaptively over Online Data Streams, in: Proc. SIGKDD'2003

[10] G.Cormode, F.Korn, S.Muthukrishnan, and Divesh Srivastava, Finding Hierarchical Heavy Hitters in Data Streams, in: Proc. VLDB'2003

[11] J.Cheng, Y.Ke, and W.Ng, A Survey on Algorithms for Mining Frequent Itemsets over Data Streams, Knowledge and Information Systems, 16 (1) (2006) 1-27

[12] G.Cormode, and S.Muthukrishnan, What's Hot and What's Not:Tracking Most Frequent Items Dynamically, in: Proc. PODS'2003

[13] C.Giannella, J.Han, J.Pei, X.Yan, and P.Yu, Mining Frequent Patterns in Data Streams at Multiple Time Granularities, in: Proc. AAAI/MIT'2003

[14] J.Han, H.Cheng, D.Xin, and X.Yan, Frequent pattern mining: current status and future directions, Data Mining and Knowledge Discovery, 17 (2007) 55-86

[15] J.Han, J.Pei, Y.Yin, and R.Mao, Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach, in: DMKD'2004

[16] R.Jin, and G.Agrawal, An Algorithm for In-Core Frequent Itemset Mining on Streaming Data, in: Proc.

ICDM'2005

[17] C.Jin, W.Qian, C.Sha, J.X.Yu, and A.Zhou, Dynamically Maintaining Frequent Items Over A Data Stream, in: Proc. CIKM'2003

[18] S.Kevin, and R.Ramakrishnan, Bottom-Up Computation of Sparse and Iceberg CUBEs, in: Proc. SIGMOD'1999.

[19] J.Koh, and S.Shin, An Approximate Approach for Mining Recently Frequent Itemsets from Data Streams, in: Proc. DaWak'2006

[20] C.K.Leung, and Q.I.Khan, DSTree: A Tree Structure for the Mining of Frequent Sets from Data Streams, in: Proc. ICDM'2006

[21] D.Lee, and W.Lee, Finding Maximal Frequent Itemsets over Online Data Streams Adaptively, in: Proc. ICDM'2005

[22] G.S.Manku, and R.Motwani, Approximate Frequency Counts over Streaming Data, in: Proc. VLDB'2002

[23] B.Mozafari, H.Thakkar, and C.Zaniolo, Verifying and Mining Frequent Patterns from Large Windows over Data Streams, in: Proc. ICDE'2008

[24] G.Mao, X.Wu, X.Zhu, and G.Chen, Mining Maximal Frequent Itemsets from Data Streams, Journal of Information Science 33 (3) (2007) 251-262

[25] N.Pasquier, Y.Bastide, R.Taouil, and L.Lakhal, Discovering frequent closed itemsets for association rules, in: Proc. ICDT'1999

[26] B.Padmanabhan, and A.Tuzhilin, On Characterization and Discovery of Minimal Unexpected Patterns in Rule Discovery, IEEE Transactions on Knowledge and Data Engineering 18 (2) (2006) 202-216

[27] W. Teng,M.Chen, and P.S.Yu, A Regression-Based Temporal Pattern Mining Scheme for Data Streams, in: Proc. VLDB'2003

[28] F.Tao, F.Murtagh, and M.Farid, Weighted Association Rule Mining using Weighted Support and Significance Framework, in: Proc. SIGKDD'2003

[29] H.J.Woo, and W.S.Lee, estMax: Tracing Maximal Frequent Itemsets over Online Data Streams, in: Proc. ICDM'2007

[30] H.Li, S.Lee, and M.Shan. Online Mining(Recently) Maximal Frequent Itemsets over Data Streams, in: Proc. RIDE'2005

[31] K.Gouda and M.J.Zaki. Efficiently Mining Maximal Frequent Itemsets, in Proc.ICDM'2001.

[32] R.J.Bayardo. Efficiently Mining Long Patterns from Databases, in Proc.SIGMOD'1998.

[33] R.C.Agarwal, C.C.Aggarwal and V.V.V.Prasad. Depth First Generation of Long Patterns, in Proc.SIGKDD'2000.

[34] D.Burdick, M.Calimlim and J.Gehrke. MAFIA: A Maximal Frequent Itemsets Algorithm for Transactional Databases, in Proc. ICDE'2001.

[35] G.Yang. The Complexity of Mining Maximal Frequent Itemsets and Maximal Frequent Patterns. in Proc. SIGKDD'2004.