Mining Maximal Frequent Itemsets in Data Streams Based on FP-Tree

Fujiang Ao[1], Yuejin Yan[2], Jian Huang[1], and Kedi Huang[1]

[1] School of Mechanical Engineering and Automation, National University of Defense Technology, Changsha, 410073, China
[2] School of Computer Science, National University of Defense Technology, Changsha, 410073, China
fjao@nudt.edu.cn

Abstract. Mining maximal frequent itemsets in data streams is more difficult than mining them in static databases for the huge, high-speed and continuous characteristics of data streams. In this paper, we propose a novel one-pass algorithm called FpMFI-DS, which mines all maximal frequent itemsets in Landmark windows or Sliding windows in data streams based on FP-Tree. A new structure of FP-Tree is designed for storing all transactions in Landmark windows or Sliding windows in data streams. To improve the efficiency of the algorithm, a new pruning technique, extension support equivalency pruning (ESEquivPS), is imported to it. The experiments show that our algorithm is efficient and scalable. It is suitable for mining MFIs both in static database and in data streams.

Keywords: maximal frequent itemsets, data streams, FP-Tree, pruning technique.

1 Introduction

In recent years, data streams have been researched widely. The technologies about data streams are used in many applications. Examples of such applications include financial applications, network monitoring, security, telecommunications data management, web applications, manufacturing, sensor networks, and others [1]. In a word, a data stream is a real-time, continuous, ordered (implicitly by arrival time or explicitly by timestamp) sequence of items. The algorithm for mining data streams must be single-pass algorithm for the characters of data streams.

The time and space efficiency of data mining in data streams is more significant than that in static databases. The number of maximal frequent itemsets and closed frequent itemsets is much less than that of frequent itemsets. So, mining MFIs or CFIs can get better time and space efficiency than mining frequent itemsets. Mining maximal frequent itemsets [2][3][4] and mining closed frequent itemsets [5][6] in data streams is to be a tendency.

Many good algorithms have been developed for mining maximal frequent itemsets in static database, for example MaxMiner [7], DepthProject [8], GenMax [9], AFOPT

[10], FPMax* [11], FpMFI [12]. All these algorithms need to scan database more than one pass. They are not suitable for mining maximal frequent itemsets in data streams. In all these algorithms, FpMFI is almost fastest for all tested database [12]. The algorithm needs to scan database two passes. We reconstruct the algorithm to a single-pass one, called FpMFI-DS. To mining maximal frequent itemsets in Landmark windows or Sliding windows in data streams, we must store all transactions in the window. For Sliding windows, when transaction is out of window, it should be deleted from window. To satisfy with these requires, we designed a new structure of FP-Tree, which can store all transactions in Landmark windows or Sliding windows, and when transaction is out of Sliding windows, it can be deleted. To reduce search space of FpMFI-DS, a new pruning technique, extension support equivalency pruning, is added in the algorithm. The efficiency of FpMFI-DS is close to FPMax* and a little lower than that of FpMFI.

2 Preliminaries and Related Work

This section will formally describe the MFIs mining problem in data streams and the set enumeration tree that represents search space. Also the related works will be introduced in this section.

2.1 Problem Revisit

Let $I = \{i_1, i_2, ..., i_m\}$ be a set of *m* distinct elements, called *items*. A subset $X \subseteq I$ is called an *itemset*. An itemset with *k* items is called a *k*-itemset. Each transaction *t* is a set of items in *I*. A data stream, $DS = [t_1, t_2, ...t_N)$, is an infinite sequence of transaction. For all transactions in a given window *W* over data stream, the support of an itemset *X*, denoted as *sup(X)*= $D_x / |W|$, where D_x is the number of transactions in which *X* occurs as a subset and $|W|$ is the width of the window. For a given threshold *min_sup* in the range of [0,1], itemset *X* is frequent if *sup(X)* $\geq$ *min_sup*. If *sup(X)* $\geq$ *min_sup* and for any $Y \supseteq X$, we have *sup(Y)* $<$ *min_sup*, then *X* is called maximal frequent itemset in window *W*.

From the definitions above, we can see that the selection of window *W* is important for an itemset *X* be a frequent one. In paper [13], three windows models are introduced, including landmark windows, sliding windows, damped windows. In this paper, we focus on mining the set of all maximal frequent itemsets in landmark windows or in sliding windows over data streams.

To get all maximal frequent itemsets, one method is to enumerate all itemsets that maybe be maximal frequent itemsets, count the support of these itemsets and decide whether they are maximal frequent itemsets. In paper [14], Rymon presents the concept of generic set enumeration tree search framework. The enumeration tree is a virtual tree. It is just used to illustrate how sets of items are to be completely enumerated in a search problem. The tree could be traversed depth-first, breadth-first, or even best-first as directed by some heuristic. In the domain of data mining, the set enumeration tree is also named after search space tree.

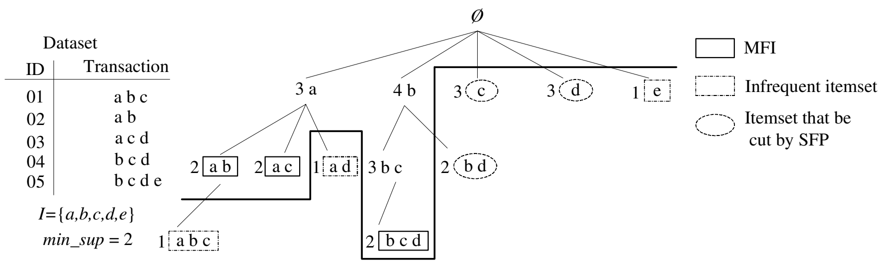
But, when the number of different items is big, the algorithm that searches all search space may suffer from the problem of combinatorial explosion. So the key to an efficient set-enumeration search is the pruning techniques that are applied to remove entire branches from consideration [7]. The two most often used pruning techniques, subset infrequency pruning and superset frequency pruning, are based on following two lemmas:

Lemma 1. A restricted subset of any frequent itemset is not a maximal frequent itemset.

Lemma 2. A subset of any frequent itemset is a frequent itemset, and a superset of any infrequent itemset is not a frequent itemset.

For example, for the dataset in the left, Fig. 1 shows the corresponding search space tree. In Fig. 1, we suppose $I = \{a,b,c,d,e\}$ is sorted in firm lexicographic order. The pruning techniques used in the tree includes *subset infrequency pruning* (SIP) and *superset frequency pruning* (SFP). The root of the tree represents the empty itemset, and the nodes at level k contain the k-itemsets. The itemset associated with each node, n, will be referred as the node's *head*(n). The possible extensions of the itemset is denoted as *con_tail*(n), which is the set of items after the last item of *head*(n). The frequent extensions denoted as *fre_tail*(n) is the set of items that can be appended to *head*(n) to build the longer frequent itemsets. In depth-first traversal of the tree, *fre_tail*(n) contains only the frequent extensions of n. The itemset associated with each children node of node n is build by appended one of *fre_tail(n)* to *head (n)*. As example in Fig. 1, suppose node n is associated with $\{b\}$, then *head*(n) = $\{b\}$ and *con_tail(n)* = $\{c,d,e\}$. For $\{e\}$ is not frequent, *fre_tail*(n) = $\{c,d\}$. The children node of n, $\{b,c\}$, is build by appending c from *fre_tail*(n) to $\{b\}$.

The problem of MFI mining can be thought as to find a border of the tree, all the elements above the border are frequent itemsets, and others are not. All MFIs are near the border. As our examples in Fig. 1, itemsets in solid rectangle are MFIs.


Dataset

ID	Transaction
01	a b c
02	a b
03	a c d
04	b c d
05	b c d e

Fig. 1. The set enumeration tree built based on the dataset in the left

2.2 Related Work

Some one-pass algorithms for mining maximal frequent itemsets in data streams have been developed, for example, DSM-MFI [3], estDec+ [4] and INSTANT [2]. They are all approximate algorithm.

DSM-MFI mines the set of all maximal frequent itemsets in landmark windows over data streams. The algorithm is composed of four steps. First, it reads a window of transactions from the buffer in main memory, and sorts the items of transactions in a lexicographical order. Second, it constructs and maintains the in-memory summary data structure, *SFI-forest.* Third, it prunes the infrequent information from the summary data structure. Fourth, it searches the maximal frequent itemsets from the current summary data structure. Steps 1 and 2 are performed in sequence for a new incoming basic window. Steps 3 and 4 are usually performed periodically or when it is needed [3]. The experiment results in paper [3] show that DSM-MFI is efficient on both sparse and dense datasets, and scalable to very long data streams.

estDec+ use a structure, CP-tree (Compressed-prefix tree), to keep the supports of all the significant itemsets in main memory. It also consists of four phases: parameter updating, node restructuring, itemset insertion, and frequent itemset selection. When a new transaction T_k in a data stream D_{k-1} is generated, these phases except the frequent itemset selection phase are performed in sequence. The frequent itemset selection phase is performed only when the up-to-date result set of frequent or maximal frequent itemsets is requested. The main advantage of the algorithm is that it adopts an adaptive memory utilization scheme to maximize the mining accuracy for confined memory space at all times [4].

INSTANT mines maximal frequent itemsequences from data streams based on a new mining theory provided by paper [2]. Where an itemsequence is an ordered list of items. The main advantage of the algorithm is that it is an online algorithm, which can directly display current maximal frequent itemsequences while they are generated. But the time efficiency of the algorithm is affected.

Paper [12] proposed a MFIs mining algorithm, FpMFI. It is an improvement over FPMax* and outperforms FPMax* by 40% averagely. They all need to scan dataset two passes. In this paper, we propose an algorithm, FpMFI-DS, based on FpMFI. FpMFI-DS only need to scan dataset one pass. It is a one-pass and exact algorithm.

3 FpMFI-DS

In this section, FpMFI-DS algorithm is introduced in details.

3.1 The Construction of FP-Tree in FpMFI-DS

To construct FP-Tree, it usually needs to scan database two passes. The first scan of database derives a list of frequent items. Then it sorts the items by frequency descending order. The list of items in header table and each path of prefix-tree will follow this order. The second scan of database gets every transaction and inserts all frequent items in transaction into FP-Tree. During the process of mining, to construct the FP-Tree of node *n*, it needs to scan the *head(n)*'s conditional pattern base that comes from FP-tree of its parent node two passes[15]. Paper [11] improves this approach by adopting an array-based technique. It only needs to scan *head(n)*'s conditional pattern base one pass.

In FpMFI-DS, to implement one-pass algorithm, we must complete the construction of FP-Tree by only scanning dataset one-pass.

To mine maximal frequent itemsets in sliding windows, the FP-Tree of root should contain all the transactions in the Sliding windows. When a transaction comes to window, all items of the transaction are inserted to the FP-Tree of root, whether they are frequent or infrequent. And when a transaction is out of window, it should be deleted from the FP-Tree of root. So except for header table and prefix-tree, the FP-Tree of root in FpMFI-DS also contains a *tidlist*, a list of IDs of the transactions in window. Every item in the *tidlist* is composed of an ID of transaction (an integer) and a pointer to the last node of the transaction in the FP-Tree of root. For an one-pass algorithm, when adding the transaction to the root FP-Tree, we can't get the frequencies of items in all transactions. So the order of the items in the FP-Tree of root can't be frequency descending order. In FpMFI-DS, the order of the items in the FP-Tree of root is based on the lexicographical order of the items. When a transaction comes to window, all items of the transaction are inserted to FP-Tree by lexicographical order. When a transaction is out of window, the last item of the transaction in the FP-Tree of root can be found through the transaction's ID and pointer in the *tidlist*, then it is deleted from root FP-Tree. To mine maximal frequent itemsets in landmark windows, we only need to fixup beginning side of the window.

The subsequent FP-Tree during the process of mining is similar to that in FPMFI. To improve the effectiveness of superset frequency pruning, the order of the items in the subsequent FP-Tree also adopts frequency descending order.

For example, for data streams and window width in the left, Fig. 2 shows the FP-tree of root built based on transactions in first window. The FP-tree includes five transactions.

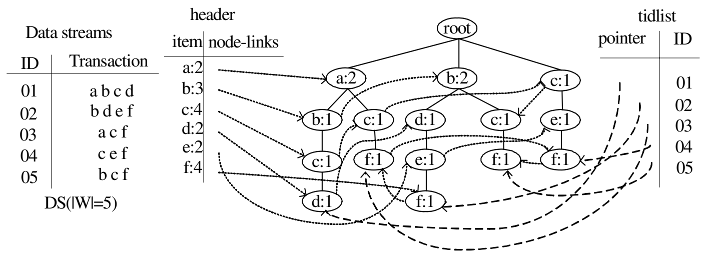


Fig. 2. The FP-Tree of root built based on transactions in first window

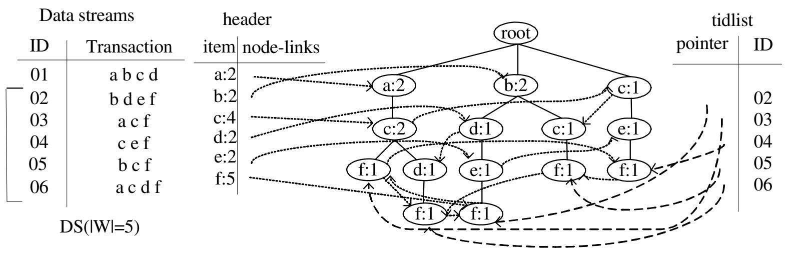


Fig. 3. The FP-Tree of root built based on transactions in second window

Fig. 3 shows the FP-tree of root built based on transactions in second window. In the FP-Tree, the first transaction is deleted from it and the sixth transaction is inserted into it.

When *min_sup* is 2, Fig. 4 shows the FP-tree of itemset {*f*} during the process of mining for data in Figure 3. The items order in Fig. 2 and Fig. 3 is based on the lexicographical order of the items, while that in Fig. 4 is based on frequency descending order.

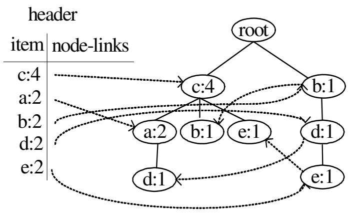


Fig. 4. The FP-Tree of itemset {*f*} during the process of mining for data in Figure 2

3.2 Pruning Techniques

FpMFI uses three pruning techniques, including subset infrequency pruning, superset frequency pruning, parent equivalence pruning. The efficiency of these pruning techniques is high for item ordering policy used by it. In FpMFI, since the item order in the FP-Tree of root is based on the lexicographical order, the item order in first level of search space tree has to accord with it. If only use these pruning techniques, the efficiency is lower than that in FpMFI, especially for dense dataset. For example, for dataset *MUSHROOM*, search space of FpMFI-DS is about as twice as that of FpMFI. So, Excepting for these pruning techniques, another pruning technique, ESEquivPS, is adopted by FpMFI-DS. ESEquivPS is firstly present in paper [16]. The pruning technique is described as following:

Supposed *p* and *n* are nodes in search space tree, and *n* is a children node of *p*. Let item $x \in fre\_tail(p)$ and $x \in fre\_tail(n)$. If $sup(head(p) \cup \{x\}) = sup(head(n) \cup \{x\})$, then any offspring node of *p* that contains item *x* and is in the right of node *n* can be pruned.

***Proof*.** Let *j* be an item associated with node *n*, *X* and *Y* are itemsets associated with $head(p) \cup \{x\}$ and $head(n) \cup \{x\}$, respectively. Since $sup(X) = sup(Y)$, then any transaction *T* containing *X* must contains item *j*. Thus, the maximal frequent itemset containing *X* must containing *j*.In the *p*-subtree, the itemsets, which associated with the nodes that contain item *x* and are in the right of node *n*, must not contain item *j* for the character of search space tree. So, they can't be maximal frequent itemsets.

From experiments, we found that if the nodes in every level of search space tree is in the order of frequency descending, then the pruning technique is invalidity. Fortunately, the first level of search space tree in FpMFI-DS is in the lexicographical order of the items. Then we can use it for the first level of search space tree. The experiments show that for the dense datasets, *MUSHROOM*, the size of search space can be trimmed off by about 30%.

3.3 Algorithm FpMFI-DS

Fig. 5 shows algorithm FpMFI-DS. Though the items order in the first level of search space tree of FpMFI-DS is different from that of FpMFI, the mining procedure of the two algorithms is similar. The difference is that algorithm FpMFI-DS adopts the new pruning technique, ESEquivPS (line 4 to line 6).

```
PROCEDURE: FpMFI-DS Algorithm
INPUT:
   n: a node in search space tree that associated with a head itemset h,
      a FP-tree, a MFI-tree, and an array
   M-trees: MFI-trees of all ancestor nodes of n
1  For each item x from end to beginning in header of n.FP-tree
2    h'=h ∪ {x} //h' identifies n'
3    if (sup(h')<min_sup)  continue
4    if (ESEquivPS_cheching(x))  continue
5    if (Thirdlevel() and sup({x})= = sup(h'))
6      insert true into respective position of a bool array for ESEquivPS
7    if x is not the end item of the header
8      if(superset_checking(con_tail(n'),n.MFI-tree)  return
9      if(superset_checking(con_tail(n'),n.FP-tree)
10       insert h' ∪ con_tail(n') into M-trees return
11    if n.array is not null
12       fre_tail(n') = {frequent items for x in n.array}
13    else
14      fre_tail(n') = {frequent items in conditional pattern base of h' }
15    PeIs = {items whose count equal to the support of h'}
16    if(superset_checking(fre_tail(n'), n. MFI-tree)
17      if the number of items before x in the header is |fre_tail(n')|
18          return
19     else continue
20    if(superset_checking(fre_tail(n'), n. FP-tree)
21      insert h' ∪ fre_tail(n') into M-trees
22      if the number of items before x in the header is |fre_tail(n')|
23         return
24      insert fre_tail(n') into n.MFI-tree continue
25    h' = h' ∪ PeIs , fre_tail(n') = fre_tail(n') – PeIs
26    sort the items in fre_tail(n')
27    construct the FP-tree of n'
28    if(superset_checking(fre_tail(n'), n'. FP-tree)
29     insert h' ∪ fre_tail(n') into M-trees
30     if the number of items before x in theheader is |fre_tail(n')|
31        return
32     insert fre_tail(n') into n.MFI-tree continue
33    construct the MFI-tree of n'
34    M-trees = M-trees ∪ {n.MFI-tree}
35    call FpMFI-DS(n' ,M-trees)
```

Fig. 5. Algorithm FpMFI-DS

To implement ESEquivPS, we use an integer array store the support of items in first level of search space tree and use a bool array denote if the respective items satisfy with the condition of ESEquivPS. When exploring the third level of search space tree, we check if the support of respective item equals to that of the corresponding item in the first level of search space tree. If they are, the corresponding position in the bool array is set *true* (line 5 to line 6). Before exploring any node in the search space tree, we first chech if the corresponding position in the bool array is set to *true*. If it is, the node should be cut off (line 4).

4 Experimental Evaluations

All the experiments were conducted with a 2.4 GHZ Pentium IV with 512 MB of DDR memory running a Redhat Linux 9.0 operation system. We implemented the code of FpMFI-DS by c++ and compiled it with the g++ 2.96 compiler.

4.1 Performance Comparisons

To evaluate the performance of FpMFI-DS, we have compared its performance with a representative algorithm, INSTANT [2]. The advantage of INSTANT is that it can directly display current maximal frequent itemsequence (not itemsets) while they are generated. For sparse datasets, the efficiency of the algorithm is high. But for dense datasets, the efficiency is not very good. The code of INSTANT was provided by its authors, Guojun Mao, etc. It is also written in c++ and compiled by g++ 2.96 compiler.

The dataset in the experiment is *T20I5D10K*, a dataset generated by IBM data generator [17]. The synthetic dataset *T20I5D10K* has average transaction size *T* of 20 items and the average size of frequent itemset *I* of 5 items and the number of transactions *D* of 10K. It is a sparse dataset.

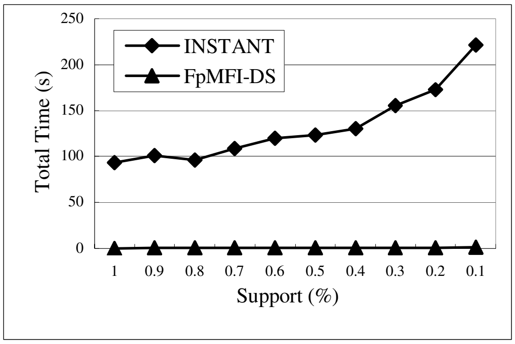


Fig. 6. Performance comparisons with INSTANT

Fig. 6 shows the result of performance comparisons for dataset *T20I5D10K*. The efficiency of FpMFI-DS is much higher than that of INSTANT. For the dataset, maximal total time of FpMFI-DS is lower than 2 seconds.

We also compared its performance with some multi-pass algorithms. Fig. 7 and Fig. 8 show the result of performance comparisons with algorithm FPMax*. The dataset in Fig. 7 is *T20I5D100K*, a sparse dataset and the dataset in Fig. 7 is

MUSHROOM, a dense dataset. The source code of algorithm FPMax* and dataset *MUSHROOM* were downloaded from [18]. Algorithm FPMax* is written in c++ and compiled with the g++ 2.96 compiler, too.

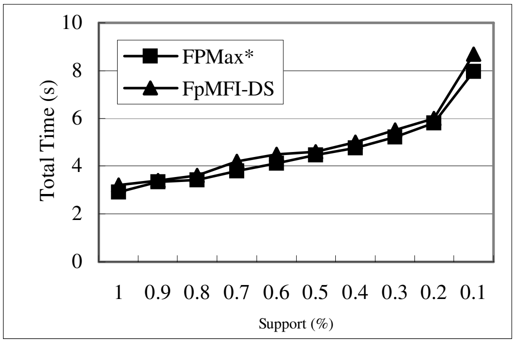


Fig. 7. Performance comparisons with FPMax* for dataset *T20I5D100K*

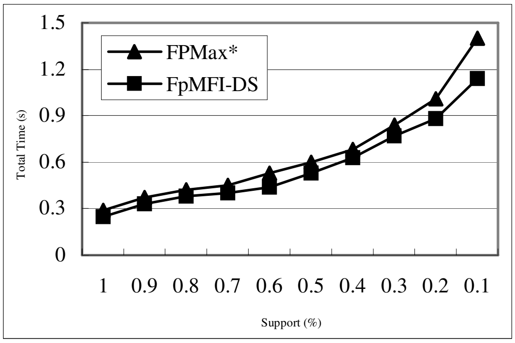


Fig. 8. Performance comparisons with FPMax* for dataset *MUSHROOM*

From Fig. 7 and Fig. 8, we can see that for dense dataset, the efficiency of FpMFI-DS is a little higher than that of FPMax*, and for sparse dataset, the efficiency of FpMFI-DS is a little lower than that of FPMax*. We can draw a conclusion that the efficiency of two algorithms is close. The result in paper [11] shows that the efficiency of FPMax* is very high. So the efficiency of FpMFI-DS is good, too.

4.2 Scalability of FpMFI-DS

To evaluate the scalability of FpMFI-DS, we use four huge datasets, *T10I5D1000K*, *T10I5D2000K*, *T10I5D3000K*, *T10I5D4000K*. The minimum support is 0.1%. From Fig. 9, we can see that the execution time grows smoothly as the dataset size increases from 1,000K to 4,000K. The algorithm has scalability.

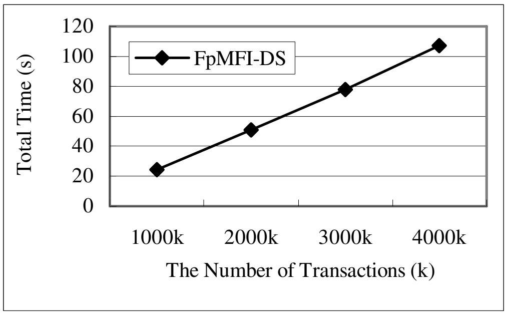


Fig. 9. The scalability of FpMFI-DS

5 Conclusions

In this paper, we proposed a novel one-pass algorithm, FpMFI-DS, which mines all set of the maximal frequent itemsets in data streams. To mine MFIs both in landmark windows and in sliding windows, we adopt a new structure of FP-Tree. To reduce the search space of the algorithm, a new pruning technique, ESEquivPS, is adopted by the algorithm. The experiments show that the algorithm is efficient on both sparse and dense datasets, and has good scalability.

Acknowledgements

We would like to thank Guojun Mao for providing the code of algorithm INSTANT. The work of this paper was supported by the Natural Science Foundation of China under the Grant No. 60573057.

References

1. Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: Models and issues in data stream systems. In: Proc. of the twenty-first ACM SIGMOD-SIGACTSIGART Symposium on Principles of Database Systems 2002, pp. 1–16 (2002)
2. Mao, G., Wu, X., Liu, C.: Online Mining of Maximal Frequent Itemsequences from Data Streams. University of Vermont, Computer Science Technical Report, CS-05-07 (2005)
3. Li, H., Lee, S., Shan, M.: Online mining (recently) maximal frequent itemsets over data streams. In: Proc. of the fifteenth International Workshops on Research Issues in Data Engineering: Stream Data Mining and Applications, Tokyo, Japan, pp. 11–18. IEEE Press, NJ (2005)
4. Lee, D., Lee, W.: Finding maximal frequent itemsets over online data streams adaptively. In: Proc. of the Fifth IEEE International Conference on Data Mining.Houston, USA, pp. 266–273. IEEE Press, NJ (2005)
5. Chi, Y., Wang, H., Yu, P S, Muntz, R.: Moment: maintaining closed frequent itemsets over a stream sliding window. In: Proc. of the fourth IEEE International Conference on Data Mining, UK, pp. 59–66. IEEE Press, NJ (2004)

6. Jiang, N., Gruenwald, L.: CFI-Stream: mining closed frequent itemsets in data streams. In: Proc. of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, Philadelphia, PA, USA, 2006, pp. 592–597 (2006)
7. Bayardo, R.: Efficiently mining long patterns from databases. In: ACM SIGMOD Conference (1998)
8. Agarwal, R., Aggarwal, C., Prasad, V.: A tree projection algorithm for generation of frequent itemsets. Journal of Parallel and Distributed Computing (2001)
9. Gouda, K., Zaki, M.J.: Efficiently Mining Maximal Frequent Itemsets. In: Proc. of the IEEE Int. Conference on Data Mining, San Jose (2001)
10. Rigoutsos, L., Floratos, A.: Combinatorial pattern discovery in biological sequences: The Teiresias algorithm. Bioinformatics 14(1), 55–67 (1998)
11. Grahne, G., Zhu, J.: Efficiently Using Prefix-trees in Mining Frequent Itemsets. In: Proc. of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, November 19, 2003, Melbourne, Florida, USA (2003)
12. Yan, Y., Li, Z., Chen, H.: Fast Mining Maximal Frequent ItemSets Based on FP-Tree. In: Webb, G.I., Yu, X. (eds.) AI 2004. LNCS (LNAI), vol. 3339, pp. 475–487. Springer, Heidelberg (2004)
13. Zhu, Y., Shasha, D.: StatStream: Statistical monitoring of thousands of data streams in real time. In: Bernstein, P., Ioannidis, Y., Ramakrishnan, R. (eds.) Proc. of the 28th Int'l Conf. on Very Large Data Bases, Hong Kong, pp. 358–369. Morgan Kaufmann, Seattle (2002)
14. Rymon, R.: Search through Systematic Set Enumeration. In: Proc. of Third Int'l Conf. on Principles of Knowledge Representation and Reasoning, pp. 539–550 (1992)
15. Han, J., Pei, J., Yin, Y.: Mining Frequent Patterns without Candidate Generation. In: Proc. 2000 ACM-SIGMOD Int. Conf. on Management of Data (SIGMOD'00), May 2000, Dallas, TX (2000)
16. Ma, Z., Chen, X., Wang, X.: Pruning strategy for mining maximal frequent itemsets. Journal of Tsinghua Univ 45(S1), 1748–1752 (2005)
17. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: Proc. Of the 20th Intl. Conf. on Very Large Databases (VLDB'94), Santiago, Chile, Sept, 1994, pp. 487–499 (1994)
18. Codes and datasets available at http://fimi.cs.helsinki.fi/