TELKOMNIKA, Vol. 11, No. 7, July 2013, pp. 3759 ~ 3766
e-ISSN: 2087-278X

An Efficient Algorithm for Mining Top-K Closed Frequent Item sets over Data Streams over Data Streams

Mao Yimin*[1], Xue Xiaofang[2], Chen Jinqing[1]
[1]School of Applied Science, Jiangxi University of Science and Technology, Gan Zhou, Jiangxi, 0797-8312636
[2]Chong Qing Communication Institute, Chong Qing, China, 023-84178972
Corresponding author, e-mail: mymlyc@163.com[1], xxf123@163.com[2]

Abstract

Focusing on problems such as complexities existing in compressed storage structures of the current data stream Top-k closed frequent item sets algorithm and inaccuracy in the algorithm, the paper puts forward an algorithm of MTKCFI-SW by designing compact prefix pattern trees for compression and storage of effective information in data stream sliding windows. The CFP-tree, capable of promptly capturing newly added data stream information under circumstances of any sliding window sizes, does not need to fix the sizes of sliding windows and thus improves the flexibility of this algorithm. Research in dynamic determination of mining threshold and pruning threshold also helps to improve accuracy of this algorithm by adopting an effective approach in mining Top-k closed frequent item sets in the environment of data stream.

Keywords*: data streams, Top-K closed frequent item sets, sliding window, data mining*

1. Introduction

In recent years, mining of data streams has becomes one of the most important research issues of data mining. Data streams are found in a lot of applications including network monitoring and traffic management, sensor network monitoring, transaction analysis of large electronic-commerce, stock tickler monitoring, web click steam monitoring and mining. However, new challenges have emerged. Due to their vast sizes, some stream types should be mined fast before being deleted forever. Generally, the alphabet is too large to keep exact information for all elements. Conventional database and mining techniques, though effective with stored data, are deemed impractical in this setting.

Efficient stream mining methods based on a user-specified minimum support threshold *min_sup* have been studied extensively. However, the setting of minimum support threshold is quite tricky and it leads to the following problem that may hinder its popular use. There are two challenges of minimum support based stream data mining: (1) if the value of minimum support threshold is set to be too small, the pattern mining algorithm may lead to the generation of thousands of patterns; (2) if the value of minimum support constraint is set to be too big, the mining algorithm may often generate a few patterns or even no answers. As it is difficult to predict how many patterns will be mined with a user-defined minimum support threshold, the top-K pattern mining has been proposed.

Top-K frequent item sets mining on definite database were studied in [1, 6], but they did not have counterparts in the sliding window model. Golab [2] presented an algorithm FREQUENT for detecting the Top-K frequent items in sliding windows defined over packet streams. It works in the jumping window model and performs well with bursty TPC/IP streams containing a small set of popular item types, but it deals with only one-item while cannot be employed in multiply item sets situations. Closed frequent item sets mining over data streams were studied in [7, 8, 3], and algorithms for frequent item sets mining in sliding windows were presented in [4, 9]. However they all do not resolve the basic problems of setting a support threshold. To the best of our knowledge, document 5 is little progress made thus far to explicitly address the problem of Top-K closed frequent item sets mining in sliding windows. Three

Received January 26, 2013; Revised April 15, 2013; Accepted April 28, 2013

optimal strategies are exploited to reduce time and space consumption of the algorithm: (1) pruning impotent nodes in the newest basic window whenever a sliding window updates, (2) promoting support threshold during mining process dynamically, and (3) adjusting potency parameter adaptively. But it exists three problems in the algorithm: (1) fixing the size of the sliding window, (2) generating too much price to maintaining nodes, (3) not being enough precise to define mining threshold and pruning threshold.

Our contribution is devising efficient algorithms for finding the Top-K closed item sets within the current sliding window. In this paper, three strategies are presented to reduce the mining time and space consumption dramatically, and experimental study validates their effectiveness and efficiency.

2. Preliminaries

A data stream, *DS* = [W1, W2, …, W*N*], is an infinite sequence of basic windows, where each basic window W*i*, $\forall$ i = 1, 2, …, N, is associated with a window identifier i, and N is the window identifier of the "latest" basic window WN. A basic window consists of a fixed sized number of transactions denoted by <T_1, T_2,…, T_k,…>. Eeach transaction is composed of a set of items (named item set) denoted by X_i(i=1, 2,…, p). The size of a basic window W_i is denoted by $|W_i|$.

Because it is unrealistic to store all the data into limited main memory or even in secondary storage, the single-pass algorithm for mining data streams has to sacrifice the correctness of their analytical results by allowing some frequency errors. Therefore, the true support of an item set X_i is the number of transactions of the stream containing the item set as a subset, and denoted by $f_{sw}^{'}(x_i)$. The estimated support of an item set X_i is the estimated true support stored in the summary data structure, and denoted by $f_{sw}(x_i)$. Note that $1 \le f_{sw}(x_i) \le f_{sw}^{'}(x_i)$. In this paper, the item sets embedded in the data streams can be divided into three types: frequent item set, significant item set, and infrequent item set. An item set X_i is called frequent if $f_{sw}(x_i) > (s-\varepsilon)|N|$, where *s* is a user-defined minimum support threshold in the range of [0, 1], ε is a user specified maximum support error threshold in the range of [0, *s*] and |N| is the number of transactions in sliding window. An item set X_i is called significant if $f_{sw}(x_i) > \varepsilon |N|$. An item set X_i is called infrequent if $f_{sw}(x_i) \le \varepsilon |N|$.

$X \prec Y$ denotes that lexicographical-order of item X is lower than that the item Y in this paper.

3. Algorithm Implement

3.1. The Structure of CFP-Tree

The current frequent pattern tree is an improved FP-tree [10] to adapt for incrementally mining the current frequent patterns over an online data stream. Compared with an FP-tree, a CFP-tree has the following improvements:

(1) All items in the CFP-tree are sorted in lexicographical-order. In contrast, the frequent items in the FP-tree are sorted in their frequency descending order.

(2) Each node in an FP-tree consists of three fields: item-name, count, and node-link. But in a CFP-tree, besides three fields, another field, item-sign (also means the sign of item) registers the type of node, which 0 is frequent item, 1 is significant item and 2 is infrequent item.

(3) An item-list table is used to index the CFP-tree. All items of a data stream are maintained in the table and sorted in lexicographical-order. Each entry in the table consists of four fields, item-name, count, item-sign and head of node-link, where head of node-link is a pointer pointing to the first node in the CFP-tree carrying the item-name.

(4) Each entry in the tid-list consists of two fields: tid and pointer. Tid registers the number of transaction. Pointer is a pointer pointing the last item of each branch in the CFP-tree.

3.2. Mining Algorithm of Top-k Closed Frequent Item sets in Sliding Windows

For the sake of effectively mining Top-k closed frequent item sets in sliding windows of data stream, data steam information of the current windows is initially stored in CFP-tree by

algorithm and arranged by descending order in terms of the support of each item in Item-list. The minimum support threshold value is set in accordance with Theorem 1. Subsequently, the current windows are filtered in order to reduce the maintenance cost of CFP-tree by deleting numerous infrequent items from CFP-tree. Ultimately, the mining of Top-k closed frequent item sets in CFP-tree can be conducted. Detailed procedures are illustrated as shown in Algorithm 1.

Each item in Item-list is arranged by descending order in terms of its support. Let us suppose X_k is the *Kth* item in terms of support by descending order, in which $X_i \in$ *Item-list* (I = 1, 2, …, k), $Sup_{sw}(X_k)$ is the minimum support of item X_k.

Theorem 1 In the CFP-tree structure, if $\forall x_i \in Item-list$, $Sup_{sw}(x_i) \geq Sup_{sw}(x_k)$, then the number of closed frequent item sets produced by the item *Xi* is larger than that of Top-k closed frequent item sets.

Proof the closed frequent item sets produced by item *Xi* can be classified into two cases:

(1) ∵ item $x_i \subset CFI$ ∴ $Sup_{sw}(Xi) \geq Sup_{sw}(X_k)$

From the definition of closed frequent item sets, it can be inferred that *Xi* might produce item sets:

$$x_i \cup x_j \subset CFI(x_j \in Item-list,\ j = 1, 2, ..., k) \quad Sup_{sw}(x_i \cup x_j) \geq Sup_{sw}(x_k)$$

That is to say, the number of closed frequent item sets produced by X_i is equivalent to that of the item sets X_i and $X_i \cup X_l$.

(2) item $x_i \cup x_l \subset CFI$, and $Sup_{sw}(x_i \cup x_l) \geq Sup_{sw}(x_k)$ $(x_l \in Item-list_{\ l = 1, 2, ...k})$

From the definition of closed frequent item sets, it can be inferred that item set $X_i \cup X_l$ might produce item sets:

$$x_i \cup x_l \cup x_p \subset CFI(x_p \in Item-list,\ p = 1, 2, ..., k)$$

That is to say, the number of closed frequent item sets produced by item $X_i \cup X_l$ is equivalent to that of the item sets $X_i \cup X_l$ and $X_i \cup X_l \cup X_p$.

As can be inferred from Theorem 1, in MTKCFI-SW algorithm, the minimum support of the current sliding window is determined by the minimum support of the k*th* item X_k in Item-list, which is referred to as "*Dynamic Determination of Mining Threshold*" in the paper. In this way, the mining threshold and pruning threshold can be immediately determined, which are exempted from the influence of any parameters and can fairly increase precision of the algorithm.

Algorithm 1 MTKCFI-SW

Input: the current data stream transaction
Output: Top-k closed frequent item sets in current sliding window

```
Call Updata CFP-tree( );
Call Pruning CFP-tree( );
Call Ming TPCFI( );
```

3.3. Incremental Updata

The information in sliding window over data stream is constantly changed. When new transactions in current window arrive in sliding window, the transactions in obsolete window are deleted from sliding window. Thus, the newly generated transaction can be stored immediately in the CFP-tree according to lexicographical-order as it arrives. The procedure of incrementally updating a CFP-tree is shown in algorithm 2.

Algorithm 2 Updata CFP-tree

Input: the current transaction in the data stream TC, a user-specified minimum support threshold s∈ (0,1), and a user-defined maximum support error threshold ε∈ (0,s);
Output: A CFP-tree generated so far;

```
Get the projection TC',of TC;
for each item Xi∈TC' do
 if Xi∉ Item-list then
   Create a new entry of form(xi,,1,2,node-link) into the Item-list;
 else
    Xi.count= Xi.count+1;
    Endif
 if CFP-tree has a child node with itemname such that y.itemname=xi.itemname then
   y.count=y.count+1;
```

```
  else
    Create a new entry of form (xi,,1,2,node-link) into the  CFP-tree;
  endif
endfor
Order Item-list
Min_Sup= Sup_sw(x_k) ;
for each item x_i ∈Item-list do
  if the frequent type of Xi is changed then
    set the frequent item-sign of all node with item-name=Xi in CFP-tree;
  endif
endofor
```

3.4. Efficiently Pruning CFP-tree

As data streams flow, the information of many historic transactions might become obsolete. Additionally, the number of lots of infrequent item sets is really huge. It is consumptive for a CFP-tree to maintain lots of obsolete item sets and infrequent item sets. So, an operation of pruning CFP-tree should be periodically performed to delete the obsolete and infrequent item sets.

Theorem 2. If X_i is an infrequent item set, X_i could be deleted without bringing any false error.

Proof. Suppose a sliding window, SW = [SW_1, SW_2, …, SW_n], is an infinite sequence of basic windows, where n is the number of sliding window. Additionally, |N| is the number of transaction in sliding window, |SW_i| is the number of transaction in basic window and ε is a user specified maximum support error threshold.

As any X_i in a sliding window, its support threshold is defined $f_{sw_i}(x_i)$.If X_i is infrequent item in the basic window of m and it is changed frequent item or significant item in the basic window of n-m, its true support threshold in sliding window, SW,is as following:

$$f'_{SW}(x_i) = f_{SW}(x_i) + \sum_{j=1}^{m} f_{SW_j}(x_i) \quad m<n$$

in MCFI-SW, if X_i is a infrequent item set, we can know $f_{sw_i}(x_i) \le \varepsilon \times |SW_i|$

$\sum_{j=1}^{m} f_{BW_j}(x) \le \varepsilon \times \sum_{j=1}^{m} |BW_j| < \varepsilon \times |N|$, hence, $f'_{SW}(x) - f_{SW}(x) < \varepsilon \times |N|$. ends

Theorem 3. As any path in the CFP-tree, the support threshold of nodes in the same path is descending sort and all leaf nodes represent the item set of least support threshold.

Proof. Let $node_i$ and $node_j$ be two nodes in the same path of a CFP-tree and $node_i$ be an ancestor of $node_j$.

According to the properties of the CFP-tree and algorithm 1, we can know that if $node_i$ be an ancestor of $node_j$, $node_i \prec node_j$ .Additionally, if $node_j$ is stored in the CFP-tree, $node_i$ must be inserted into the CFP-tree because $node_i$ and $node_j$ are both in the same path. But, if $node_i$ is stored in the CFP-tree, $node_j$ could not be inserted into the CFP-tree. Hence, $node_i$.count, $node_j$.count.

According to algorithm 2, we can know that all the leaf nodes are the last node inserted in each path of the CFP-tree and the support threshold of their parent nodes is greater than their support threshold. Hence, all the leaf nodes represent the item set of least support threshold. ends.

According to the properties of the CFP-tree, tid registers the number of current transaction and it corresponds to pointer which is a pointer pointing the last item of each branch in the CFP-tree. Thus, the obsolete items can be immediately moved by the pointer from the CFP-tree. At the same time, a lot of infrequent items and the items which their frequent support threshold is zero are recorded in the CFP-tree. One, if $f_{sw}(x_i) \le \varepsilon |N|, \forall x_i \in Item-list$,X_i is a infrequent item. According to theorem 3, the child nodes of X_i are infrequent items. Hence, According to theorem 2, all nodes carrying item-name are the same to X_i in the CFP-tree and their descendant are deleted. Two, $f_{sw}(x_i) > \varepsilon |N|, \forall x_i \in Item-list$, X_i is a frequent or significant item. According to theorem 2, all nodes carrying item-name is the same to X_i in the CFP-tree

cannot be deleted. But, if the nodes carrying X_i.count are zero in the CFP-tree, X_i is an obsolete item. According to theorem 3, their descendant nodes are obsolete items.

Based on the above propositions and analysis, given the maximum support error ε and the size, N, of a sliding window, the CFP-tree could be pruned by the procedure shown in algorithm 3 without needing to traverse the whole CFP-tree.

Algorithm 3. PruningCFP-tree

Input: a CFP-tree to be pruned, the number of current transaction in the data stream tid_C;

Output: a CFP-tree after being pruned

```
for each tid< tid_c, where tid ∈Tid-list do
 for each X_i ∈path which tid corresponds to pointer pointing each branch in the CFP-tree
     X_i.count= X_i.count-1
  endfor
endfor
for each X_i ∈Item-list do
  count X_i.count
  if the frequent type of X_i is changed then
    set the frequent item-sign of all node with item-name=X_i in the CFP-tree
  endif
  if f_SW( x_i) <ε*|N|
    delete all nodes carrying item-name=X_i in CFP-tree and their descent;
    delete X_i from Item-list;
  else
   delete all nodes carrying item-name=X_i and X_i.count=0 in the CFP-tree and their descent;
  endif
endfor
```

3.5. Mining Closed Frequent Item sets from CFP-tree

This paper adopts the idea of CLOSE+ [11] which bases on the famous FP-tree in static data-base to find the closed frequent item sets in data stream over sliding window. Because the CFP-tree is similar to FP-tree in structure, but different from FP-tree in dealing with the data, it needs to improve the CLOSE+ to find the closed frequent item sets from CFP-tree.

Theorem 4: In a CFP-tree that has been pruned, the potential frequent item sets are not always the leaves.

Proof: suppose |N| is the number of transactions in a sliding window, ε is a user specified maximum support error threshold, the leaf node is $node_{leaves}$. Suppose all of the leaf nodes are significant frequent items, that to say, $f_{sw}(node_{leaves}) > \varepsilon \mid N \mid$.

In CFP-tree, if an item i_m that is sorted latter location in lexicographical-order, it means $i_1 \prec i_2 \ldots \prec i_m$, i_m is leaf node, $f_{sw}(x_m) = p$ and $p > (s - \varepsilon) \mid N \mid$.Due to algorithm 2 and theorem 3, we can know the number of i_m is p, $f_{sw}(x_m) = 1 < \varepsilon \mid N \mid$, so i_m possibly is the leaf node of p branches in CFP-tree. It conflicts with the suppose, so the theore is proved.

From the theorem 3, we can know that nodes of every branch in FP-tree and CFP-tree both are ordered by decreasing support, so it is easy to build the conditional pattern base. FP–tree stores all frequent items in data-base, however CFP-tree stores all the frequent items and potential frequent items in sliding window. Compared to CLOSE+, mining Top-K closed frequent item sets from CFP-tree has two improvements. First, the algorithm has to neglect the potential frequent items, and build the conditional pattern base in frequent items in Item-list. Second, when building conditional pattern base of some item, the conditional pattern base must only include the frequent items, must not include the potential frequent items according to theorem 4. The procedure of mining Top-K closed frequent item sets is shown in algorithm 4.

Algorithm 4 MiningTPCFP

Input: *T*: a CFP-tree;

Output: Top-K closed frequent item sets;

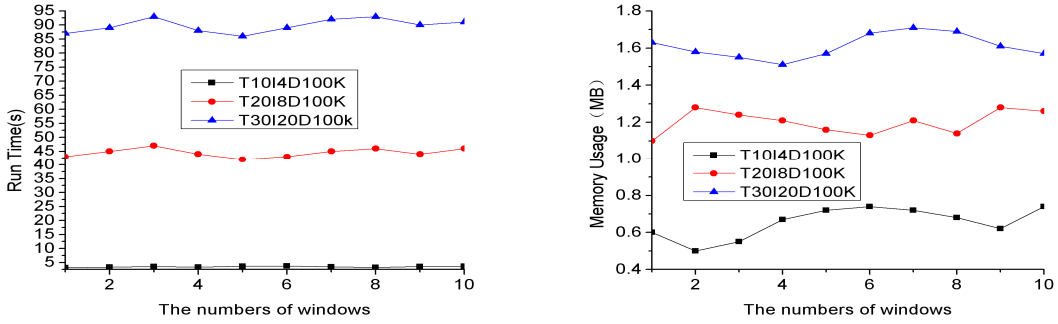
```
for each x_1 > (M in_ Sup − ε) | N | in Item-list
   call the third, fourth step in the CLOSET+ algorithm ;
endfor
traverse ITLHIR-tree,
output Top-K closed frequent item sets;
```

4. Experimental Results and Analysis

For sake of evaluating the performance of this algorithm, all experiments are conducted in Pentium IV PCs with the configurations of 2GHz in CPU, 1 GB in memory storage and Windows XP operation system. All experimental programs are compiled by C language and operated in the environment of VC ++ 6.0. The analog data in the experiment are produced by IBM analog data generators [117]. Main specifications of synthetic data are indicated as follows: *T* represent average length of transactions, *I* represents average length of frequent item sets, *D* represents the number of transactions, *1 K* represents *1000* records of transactions. All item numbers in the experiment are set by *1000*, with allowable maximum deviation by users at the value of *s/100*.

4.1. Performance Comparison of the Algorithm

Initially, the time and space efficiency of the algorithm are tested with variation of the sliding window numbers. Experiments are conducted collectively in the three data, namely, T10I4D100K, T20I8D100K and T30I20100K. In the experiment, the size of basic window in sliding window is set by 1000, the size of sliding window is set by 10000, and Top-k=100. Figure 1 (a) indicates that the consumed time in environment of numerous data categories varies slightly with the increase of transaction data stream. Figure 1(b) indicates that the storage capacity required by the algorithm in these three data varies within a small scope with the shift of sliding windows. As can be seen from Figure 1, the MTKCFI-SW algorithm has a fairly sound performance.

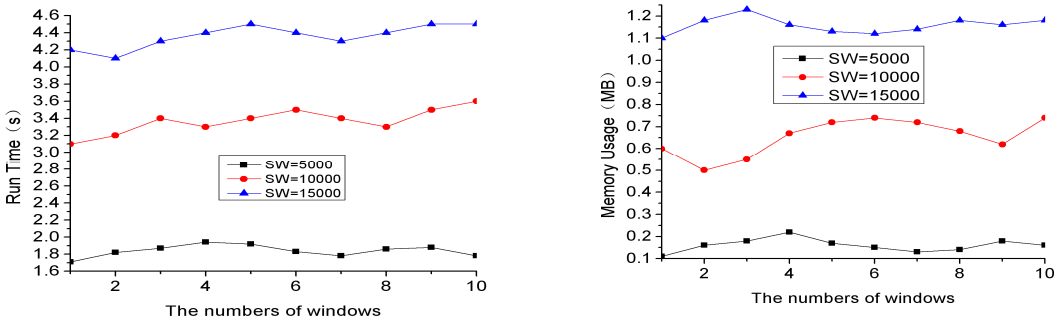


(a) Running Time of the Algorithm (b) Storage Space of the Algorithm

Figure 1. Space-time Efficiency of the Algorithm with Data Variation of Sliding Windows

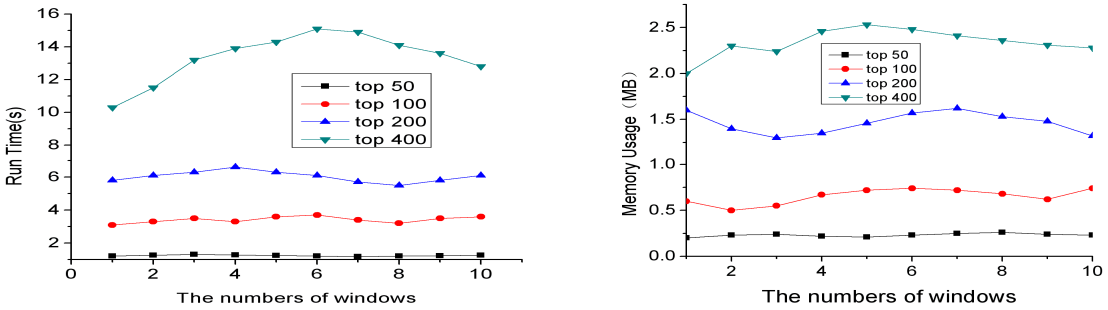
Secondly, the space-time efficiency of the algorithm can be tested with variations of the sizes of sliding windows. The experiment is conducted in the data set T10I4D100K, with the setting of Top-k=100, the settings of sizes of the sliding windows are respectively 5000, 10000 and 15000. The execution time and space consumption of ten sliding windows in the data set T10I4D100K is processed consecutively by the algorithm. As can be seen from Figure 2, with the increase of sizes of sliding windows, the consumption of resources also increases. This is because once the sizes of sliding windows increase, the information required to be maintained and searched increases as well. However, once the sizes of sliding windows are fixed, the time consumption and memory storage maintenance of the algorithm tend to be stable.

Besides, the time and space efficiency of the algorithm can be tested with variations of the sizes of Top-k. The experiment is conducted in the data set T10I4D100k, with the setting of 10000, the settings of sizes of Top-k are respectively 50, 100, 200 and 400. As can be seen from Figure 3, the consumption of time and memory storage may increase in proportion to the volume of Top-k in the process of mining. However, the time-space consumption of the algorithm tends to be stable with respect to each single sliding window.



(a) Running Time of the Algorithm (b) Storage Space of the Algorithm

Figure 2. Space-time Efficiency of the Algorithm with Size Variation of Sliding Windows

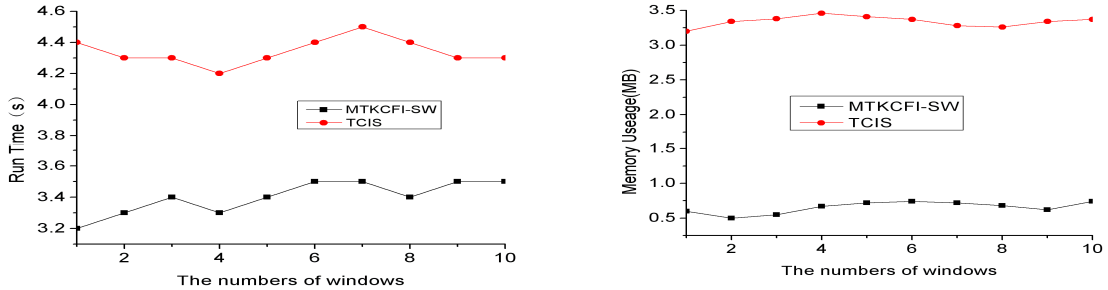


(a) Running Time of the Algorithm (b) Storage Space of the Algorithm

Figure 3. Space-time Efficiency of the Algorithm with Quantity Variation of Top-k

4.2. Performance Comparison of the Algorithm

A time-space efficiency comparison of the algorithm can be made in various sizes of sliding windows. The size of the basic window is 1000 transactions in this experiment, sizes of the sliding windows range from 10k to 100k. The experiment has adopted T10I4D100k data set to evaluate the running time and space consumption of MTKCFI-SW and TCIS when Top-k=100. Figure 4 (a) indicates that the execution time of algorithm MTKCFI-DS is less than that of TCIS; Figure 4 (b) indicates that with the increase of data, the storage capacity of MTKCFI-DS tends to be stable and the required storage capacity of algorithm MTKCFI-DS is less than that of TCIS.



(a) Running Time of the Algorithm (b) Storage Space of the Algorithm

Figure 4. Space-time Efficiency Comparison of the Algorithm with various Size of Sliding Windows

5. Conclusion

Focusing on problems such as complexities existing in compressed storage structures of the current data stream Top-k closed frequent item sets algorithm and inaccuracy in the algorithm, the chapter puts forward an algorithm of MTKCFI-SW by designing compact prefix pattern trees for compression and storage of effective information in data stream sliding windows. It improves the time efficiency of this algorithm by reducing maintenance cost, with less information storage in nodes. The CFP-tree, capable of promptly capturing newly added data stream information under circumstances of any sliding window sizes, does not need to fix the sizes of sliding windows and thus improves the flexibility of this algorithm. By adoption of pointer operations, large quantities of infrequent item sets can be deleted from the CFP-tree so as to improve time efficiency of this algorithm without traversing the whole CFP-tree. Research in dynamic determination of mining threshold and pruning threshold also helps to improve accuracy of this algorithm by adopting an effective approach in mining Top-k closed frequent item sets in the environment of data stream.

Acknowledgement

This research was supported by the research grant of department of education in Jiangxi (GJJ12347), natural science foundation of Jiangxi (20122BAB201045) and national natural science foundation of China (51164012).

References

[1] YL Cheung, AWC Fu. Mining frequent item sets without support threshold: with and without item constraints. *IEEE Transactions on Knowledge and Data Engineering*. 2004; 18(3): 1052-1069.
[2] L Golab, D Dehaan. *Identifying frequent items in sliding windows over on-line packet streams*. In SIGCOMM Internet Measurement Conference, Miami, ACM. 2003; 17(4): 173-178.
[3] X Liu, H Xu, Y Dongl. *Mining frequent closed patterns from a sliding window over data streams.* Journal of Computer Research and Development. 2006; 43(10): 1738-1743.
[4] JH Chang, WS Lee. A sliding window method for finding recently frequent item sets over online data streams. *Journal of Information Science and Engineering*. 2004; 20(2): 753-762.
[5] Yang Bei, Huang Houkuan. Mining Top-K Significant Item sets in Landmark Windows over Data streams. *Journal of Computer Research and Development*. 2010; 47(3): 463-473.
[6] AWC Fu, RWW Kwong. *Mining N-most interesting item sets.* International Symposium of Methodologies for Intelligent Systems. New York. 2000: 59-67.
[7] Y Chi, H Wang. *Moment: maintaining closed frequent item sets over a sliding window.* the Fourth IEEE International Conference on Data Mining. Houston. 2004: 59-66.
[8] N Jiang, L Gruenwald. *CFI-stream: mining closed frequent item sets in data streams*. KDD'06. Philadelphia. 2006: 592-597.
[9] C Lin, D Chiu. *Mining frequent item sets from data streams with a time-sensitive sliding window*. SIAM International Conference on Data Mining. California. 2005: 67-74.
[10] J Han, J Pei. *Mining Frequent Patterns without Candidate Generation*. In Proceeding of the ACM International Conference on Management of Data. New York. 2000: 1-12.
[11] Jianyong Wang, Han Jiawei. *CLOSET +: Searching for the Best Strategies for Mining Frequent closed Item sets*. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York. 2003: 236-245.