Sliding-Window Based Method to Discover High Utility Patterns from Data Streams

Chiranjeevi Manike and Hari Om

Abstract High utility pattern mining is one of the emerging researches in data mining. Mining these patterns from the evolving data streams is a big challenge, due the characteristics of data streams like high arrival rate, unbounded and gigantic in size, etc. Commonly there are three window models (landmark window, sliding window, time fading window) used in data streams. However, in most applications, users are interested in recent happenings. Hence, sliding window model has attracted high interest among three. Many approaches have been proposed based on the sliding window model. However, most of the approaches are based on level-wise candidate generation and text approach. In view of this, we propose an efficient one pass, tree based approach for mining high utility patterns over data streams. Experimental results show that the performance of our approach is better than the level-wise approach.

Keywords Data streams · Sliding window · High utility pattern · Frequent pattern mining · Data mining

1 Introduction

Utility pattern mining was introduced in 2003 to discover patterns based on the quantitative databases [1] Traditional frequent pattern mining was based on the support measure, where support is defined over the binary domain 0, 1, which is based on the occurrence frequency of items [2]. Frequent pattern mining considers purchased quantity of every item as 1. But in real time scenario a customer may purchase multiple items of different quantity. Let us consider a query of a sales

C. Manike (✉) · H. Om
Indian School of Mines, Dhanbad 826004, Jharkhand, India
e-mail: chiru.research@gmail.com

H. Om
e-mail: hariom.cse@ismdhanbad.ac.in

L.C. Jain et al. (eds.), *Computational Intelligence in Data Mining - Volume 3*,
Smart Innovation, Systems and Technologies 33, DOI 10.1007/978-81-322-2202-6_15

manager in a supermarket, "set of itemsets contributing more to the total profit?". To give response to such queries frequency of an itemset is not adequate measure [3]; we need to consider the purchased quantities of itemsets and their corresponding unit profits. Thus, high utility patterns are more significant in practical applications than frequent patterns. Utility pattern mining consider other useful measures of an itemset, they may be its purchased quantity, profit, cost, etc. [2]. Patterns having utility more than the user specified minimum utility threshold are called as high utility patterns. High utility pattern mining is the process of discovering patterns where the actual utility of the pattern is more than the user specified minimum utility threshold. Unlike frequent patterns, high utility patterns do not follow downward closure property [4], and it needs several utility calculations. Hence, high utility pattern mining is more complex than traditional frequent pattern mining. Liu et al. [5, 6] has observed an efficient pruning strategy among the transaction weighted utilities of itemsets, which is named as transaction weighted utility (TWU) downward closure property. Many algorithms have been developed based on the above efficient pruning strategy.

On the other hand pattern mining over data stream is a big challenge due to the evolving nature and requirements of data streams [7], as the data arriving continuously, the utilities of an itemsets may vary irregularly from one instance to other [8–10]. There are mainly three window models used in mining data streams, those are: landmark window, sliding window, and time fading window. Selecting particular window model is based on the type of knowledge to be discovered and the application domain [7, 11, 12]. Landmark window holds the information of itemsets from landmark time to current time, sliding window holds the recent information of itemsets like last 3 h, last 1,000 instances etc. Time fading window holds the information same as landmark window, in which the importance of an itemset decreases with time. Among these three models sliding window model gives the information of most recent occurred events and also many users are interested in this [13]. Especially in credit card fraud analysis, network intrusion detection, wireless sensor networks [7] where the event might last for a few milliseconds, or a few hours, or even a few days.

Unlike traditional databases, data streams do not allow algorithms to visit the data more than once. Hence, one pass algorithms are more suitable in these environments. Especially in sliding window model capturing incoming data as well as removing old data within window is main issue. In this paper, we have proposed an efficient approach, based on sliding window model, to effectively update the information of utility patterns and to efficiently discover high utility patterns.

2 Problem Definition

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a finite set of n distinct items. A data stream $DS = \{T_1, T_2, \ldots, T_m\}$ be a stream of transactions, where each transaction Ti is a subset of I. An itemset X is a finite set of items $X = \{i_1, i_2, \ldots, i_k\} \in I$, where $k(1 \leq k \leq n)$

is the length of the itemset. An itemset with length k is also called as k-itemset. Each item or an itemset in a transaction associated with a value (For example, purchased quantity) called internal utility of an itemset, denoted as $iu(i_j)$. Each item $i_j \in I$ associated with a value (profit, price, etc.) outside the transaction data stream called external utility and it is denoted as $eu(i_j)$. Definitions in this section are adapted from the previous works [2, 5].

Definition 1 Utility of an item i_j in a transaction T_i is the product of its internal utility and external utility, denoted as $u(i_j, T_i) = iu(i_j, T_i) \times eu(i_j)$. For example, utility of an item in T_1 is 12 (i.e., 2 × 6), in Tables 1 and 2.

Definition 2 Utility of an itemset X in a transaction T_i is the sum of the individual utilities of items belongs to the itemset X, $u(X, T_i) = \sum_{i_j \in X \in T_i} u(i_j, T_j)$. For example, $u(abc, T_1) = 2 \times 6 + 3 \times 4 + 2 \times 10 = 44$, in Tables 1 and 2.

Definition 3 Transaction utility of a transaction T_i is the sum of all its items utilities, denoted as tu(T_i). For example, $tu(T_2) = u(a) + u(b) + u(c) + u(e) = 52$, in Tables 1 and 2.

Definition 4 Transaction weighted utility of an itemset X is defined as the sum of all transaction utilities of transactions in which itemset X exists. For example, $twu(cd) = tu(T_1) + tu(T_5) + tu(T_6) = 195$, in Tables 1 and 2.

Definition 5 Minimum utility threshold is defined as the percentage of total transaction utility values of the data stream, defined as, $minUtil = \delta \times \sum_{T_i \in DS} tu(T_i)$.

Table 1 Example transaction data stream

Item/Tid	a	b	c	d	e	f	g
T_1	2	3	3	1	0	0	0
T_2	1	4	1	0	1	0	0
T_3	0	2	2	0	0	1	0
T_4	3	4	2	0	0	0	0
T_5	2	3	1	2	0	0	0
T_6	4	2	1	2	0	0	0
T_7	2	0	0	0	0	0	1
T_8	2	5	3	0	0	0	0

Table 2 Utility table

Item	a	b	c	d	e	f	g
Profit($)	6	4	10	15	20	30	40

Definition 6 High utility pattern mining can be defined as the process of finding set of patterns, where utilities of those patterns are more than the *minUtil*.

3 Related Works

Chan et al. [14], introduced high utility pattern mining in the year 2003. Theoretical model and basic definitions are given in [2] MEU (Mining with Expected Utility), two efficient strategies are also been introduced based on the utility and support values of itemsets. However, these strategies reduced candidate patterns effectively, it depends on the level-wise candidate generation-and-test problem and. The utility bound set by the property overestimating many patterns again filtering these huge set of patterns becomes complex task. Hence, same authors introduced two new algorithms namely UMining and UMining_H by incorporating above strategies [3]. Though, MEU, UMining, and UMining_H algorithms reduced candidate patterns, still depends on level-wise candidate generation and-test approach. Above algorithms are tried to reduce the search space using support and utility values of itemsets those are not efficient as Apriori anti-monotone property.

In 2005, Liu et al. [5, 6] has discovered an efficient heuristic pruning strategy called transaction weighted utility downward closure property. An efficient two pass algorithm was proposed by incorporating above property, which is called Two-Phase. The property says that TWU of an itemset is high then its subsets TWU values are also high. Even though, this property does not exist among the actual utilities of itemsets, it effectively used in finding potential patterns. The main drawback in this approach is this property allows some false itemsets, where the actual utilities of these itemsets are not high. Again filtering exact patterns from these potential set of patterns becomes main barrier in most of the cases. Afterwards, many approaches were proposed based on TWU downward closure property. But in data stream environment it may not possible to apply such an anti-monotone properties because the requirements of stream processing different from traditional databases. Moreover, low utility patterns may become high utility patterns when a new transaction arrives.

First approach that was proposed to mine the high utility patterns over data stream is THUI-Mine [15], called THUI-Mine. THUI-Mine is mainly integrates the tasks of Two-Phase [5, 6] algorithm for processing items in pre-processing procedure, and SWF [13] to use the concepts of filtering threshold in incremental procedure. THUI-Mine decomposed the problem of mining high utility patterns into two procedures: (1) Pre-processing procedure, (2) Incremental procedure. First, pre-processing procedure calculates the TWU of each item and discards the items whose TWU is less than the filtering threshold. Next generate the candidate 2-itemsets from the remaining items, and record their appeared partition number and

TWU value respectively. After processing each partition, candidate 2-itemsets with TWU value above filtering threshold value will be considered in next partition. Even THUI-Mine find complete set of high utility patterns, there is no pruning strategy incorporated in it to filter overestimated patterns.

To overcome the limitations in THU-Mine, two efficient algorithms were proposed in [16], called MHUI-TID and MHUI-BIT. These algorithms represent transaction information in two formats that is TIDlist and Bitvector respectively. For example, consider Table 1, in which item a encountered in transactions T_1, T_2, T_4, T5, T_6, T_7 and T_8; this can be represented in the form of Bitvector, that is ⟨11011111⟩ (assume window size is set to 8). Above information is also represented in the form of TIDlist that is {1, 2, 4, 5, 6, 7, 8}. This is a three-phase method, in first phase it builds information of items using either TIDlist or Bitvector representation. This algorithm processes the items as it was done by pre-processing procedure of THUI-Mine. In next phase it uses level-wise approach to generate the candidate itemsets of length more than 2. In second phase it builds a lexicographic tree structure to maintain the information of potential itemsets (k—itemsets, $k < 3$) and updates the information of newly identified itemsets when window slides. Third, high utility itemsets generation phase in which high utility patterns are generated based on the user queries. MHUI algorithms effectively reduced the number of potential candidates and optimal utilization of memory with these two representations. Hence, it achieved better performance over the THUI-Mine, but still need more than one database scans. MHUI keeps the information of itemsets with length 2, so remaining itemsets of length more than 2 will be processed in level-wise manner.

From the above discussion on related work of high utility pattern mining from data streams, we have identified two motivating factors to design a new algorithm which need at most one database scan and better runtime efficiency. In this paper we have designed a new one pass algorithm based on the prefix-tree structure and we did not consider the TWU values of an itemsets to disqualify items before starting the process.

4 Proposed Work

In this Section we present our proposed algorithm, in addition, in this section we briefly describe our method with example. Our method basically consists of three steps: (1) transactions processing: capturing information and storing in tree, (2) incremental: updating the information in tree when window is sliding, and (3) mining: tracing tree to produce output to the user query.

First step comprises of two subtasks, when the transactions arrives continuously in the data stream each transaction is processed one by one. After arriving transaction T_1, set of all possible patterns will be generated and utilities corresponding to

those patterns are also being calculated in parallel. Second subtask is updating all generated patterns in sliding window tree called $RHUI_{SW}$-Tree by calling a procedure namely $RHUI_{SW}$-Tree-Update. For each transaction there will be $2^n - 1$ (n is the number of items) number of patterns generated. While updating all these patterns procedure $RHUI_{SW}$-Tree-Update follows lexicographical order, because order of processing significantly affects the execution time. Patterns which are sharing prefix will be updated first that is a, ab, ac, ad, abc, acd, abcd will appear along the same path of $RHUI_{SW}$-Tree, so redundant moves for each node from root node is avoided.

While updating patterns in tree the utility value of pattern is added to total utility and also enqueue to pattern utility queue. Figure 1, represents the $RHUI_{SW}$-Tree after updating transaction T_1. Same procedure will be followed for next and subsequent transactions, until we reach the window size limit, once we exceed the limit then sliding window phase will enter. Let us assume sliding window size is set to 4, so window accommodate first four transactions T_1, T_2, T_3, and T_4. During the above process updated patterns of all transactions will also be stored in a table called PTable (Pattern Table) that is Table 3, this table will be used as lookup table while deleting the information of obsolete transactions from the sliding window. Whenever the total count of processed transactions exceeds the window size limit, $RHUI_{SW}$-Tree-IncUpdate procedure will be invoked. $RHUI_{SW}$-Tree-IncUpdate updates the patterns and corresponding utilities of T_1 from the $RHUI_{SW}$-Tree with the help of PTable. PTable helps to remove all patterns corresponding to the transaction T_1 without revisiting all nodes. Next $RHUI_{SW}$-Tree-IncUpdate invokes the $RHUI_{SW}$-Tree-Update procedure to process first transaction of next window W_2 that is T_5. For instance, in incremental phase to delete all patterns of a transaction T_1 we need to find patterns and utilities of transaction T_1, and have to visit corresponding nodes in tree to delete. The cost of first step is reduced by maintaining those patterns in PTable and cost of traversing completed tree is also minimized by maintaining node links.

Last step is tracing $RHUI_{SW}$-Tree, when user queries the system algorithm trace the tree to produce high utility patterns. To visit all nodes generally we follow either top-down or bottom-up approach, in this case both approaches are unsuitable because no need to visit all nodes. To improve the query response time at this stage, we used another table called, HTable (Table 4) to keep information of nodes whose pattern total utility is more than the minimum utility threshold. While updating patterns if pattern total utility crosses utility threshold that corresponding node like immediately stored in HTable, Table 4 shows the status of HTable after processing each transaction from T_1 to T_5.

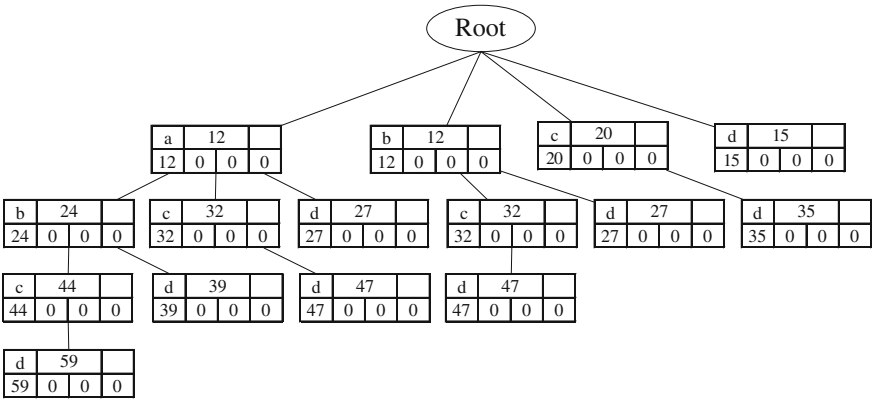

Fig. 1 $RHUI_{SW}$-Tree after updating transaction T_1

Table 3 PTable

T_1	T_2	T_3	T_4
a	a	b	a
b	b	c	b
c	c	f	c
d	e	bc	ab
…	…	…	…

Table 4 HTable

T_1	T_2	T_3	T_4	T_5
abcd	abcd	abcd	abcd	abcd
	bc	bc	bc	bc
	abc	abc	abc	abc
	abce	abce	abce	abce
		c	c	c
		…	…	…

```
Algorithm 1: RHUIMSW for mining recent high utility
             patterns
Input: A data stream DS, a pre-defined utility table, a
       user defined minimum utility threshold minUtil and
       a user specified sliding window size m
Output: Set of all high utility patterns
1 RHUI_SW-Tree ← Φ
2 Initialize sliding window size w ← m
3 PTable ← Φ
4 while a new transaction T_i arrives into DS do
5     for each transaction T_i with items n do
6           generate 2^n − 1 itemsets
7           calculate utility u of each itemset
8           while j ← 1 to 2^n − 1 do
9                 for each itemset X do
10                      call RHUI_SW-Tree-Update(X, u)
11                      store all updated nodes
                        information in PTable
12    call RHUI_SW-Tree-IncUpdate(RHUI_SW-Tree)
13    call RHUI_SW-Tree-Update(X, u)
14    store all updated nodes information in PTable
15 if user-request = true then
16    call RHUI_SW-Tree-Tracing (RHUI_SW-Tree)
17 return RHUI

Procedure 1: RHUI_SW-Tree-Update(X, u, RHUI_SW-Tree)
1 n ← size(X); X = {x_1, x_2, … x_n};
2 while l ← 1 to n do
3 search for x_i in children list of current node of
  HUI_SW-Tree;
4 if search = true then
5     case 1 search true and loop terminated, means
      patten X already exist so add utility to total
      utility and enqueue utility;
6     case 2 search true but not terminated means prefix
      itemset is already exist so continue create new
      pattern along that path;
7 else
8     make new node by creating a path from root node and
      set its utility;


Procedure 2: RHUI_SW-Tree-IncUpdate (PTable, RHUI_SW-Tree)
1 for i ← 1 to PTable.size()(i.e.,number of columns) do
2     for j ← 1 to PTable(i).size()(i.e., number of
                entries in ith column) do
3           nodelink ← PTable (i,j) delete corresponding
            pattern utility node which pointing nodelink
4 call RHUI_SW-Tree-Update(X, u, RHUI_SW-Tree)


Procedure 3: RHUI_SW-Tree-Tracing (HTable, RHUI_SW-Tree)
1 for index   1 to HTable.size() do
2     nodeline   HTable.get(index);
3     visit corresponding node of nodelink;
4     return pattern with utility;
```

5 Experimental Results

The experiments were performed on a PC with processor Intel(R) Core™ i7 2600 CPU @ 3.40 GHZ, 2 GB Memory and the operating systems is Microsoft Windows 7 32-bit. All algorithms are implemented in Java. All testing data was generated by the synthetic data generator provided by Agrawal et al. in [4]. However, the IBM generator generates only the quantity of 0 or 1 for each item in a transaction. In order to adapt the dataset into the scenario of high utility pattern mining, the quantity of each item is generated randomly ranging from 1 to 5, and profit of each item is also generated randomly ranging from 1 to 20. To fit in the real time scenario we have generated the profits by following lognormal distribution (Fig. 2). In simulation of results we have fixed the sliding window size to 1,000 for all the experiments. Different data sets are used to compare the performance of our proposed algorithm against THUI-Mine, MHUI-TID.

5.1 *Performance of RHUIM$_{SW}$ with Varying Minimum Utility Threshold*

In this section, we show the performance of our proposed algorithm RHUIM$_{SW}$ over the dataset D50KT5N1000, parameters D, T and N represents the number of transactions, average number of items per transaction and total number of distinct items respectively. We have considered minimum utility threshold range from 0.5 to 1.5 %. From Fig. 3, we can observe that the performance of all algorithms increasing with minimum utility thresholds, because when the utility threshold is high then the number of high utility patterns is very low. From Fig. 3, we can

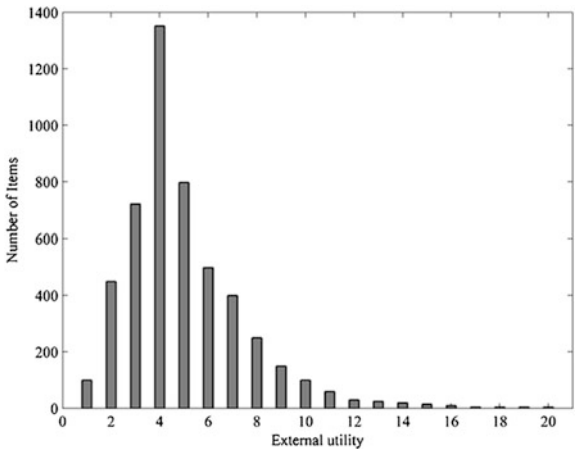

Fig. 2 Performance with varying minimum utility threshold

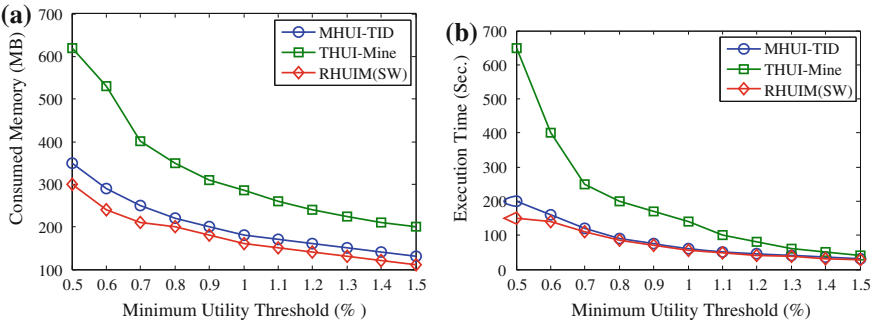

Fig. 3 Performance with varying minimum utility threshold. **a** Consumed memory. **b** Execution time

observe that proposed algorithm consumed less amount of memory than the remaining algorithms. There is some correlation in the amount of memory consumed by the MHUI-TID and proposed algorithm that we can observe in Fig. 3. The reason behind that is both the algorithms are using tree structure to maintain all the potential patterns. Even though, the execution time is very less among the three algorithms; our proposed algorithm does not give much better performance over MHUI-TID.

5.2 *Performance of RHUIM$_{SW}$ with Varying Number of Transactions*

In this section, we compare the performances of algorithms over the dataset DxKT5N1000, where x takes the values from 50 to 600. For all experiments we have set the minimum utility threshold to 0.9 %. In this experiment we have shown the scalability performance of algorithms with varying number of transactions. From Fig. 4, we can observe that the scalability of MHUI-TID and our proposed far better than the THUI-Mine algorithm.

5.3 *Performance of RHUIM$_{SW}$ with Varying Parameters*

In this section, the performance of our proposed algorithm is compared against MHUI-TID and THUI-Mine algorithms with varying average number of items per transaction and number of items. From Fig. 5, we can observe that execution time exponentially increasing with parameter especially from 6 and above values. From these results, we have understood that when the data set becomes dense the

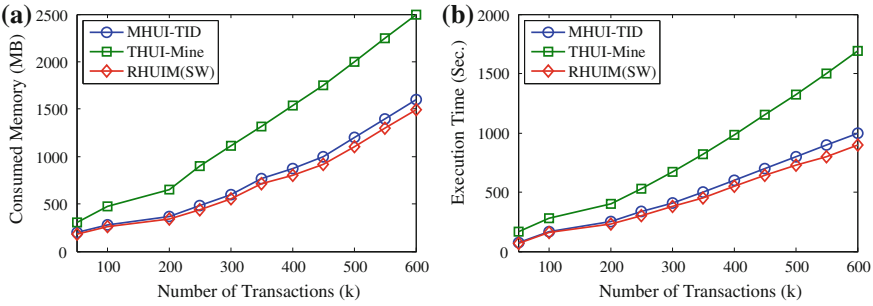

Fig. 4 Performance with varying number of transactions. **a** Consumed memory. **b** Execution time

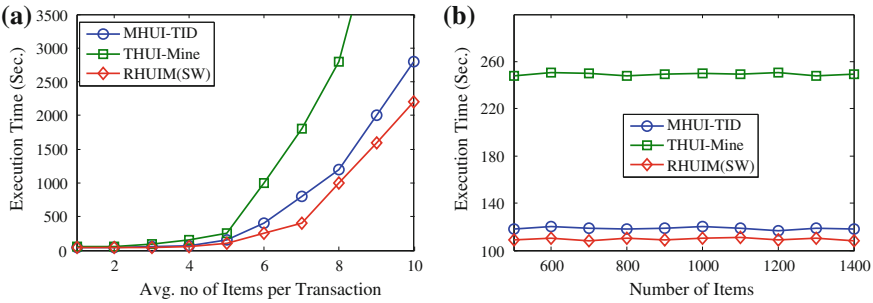

Fig. 5 Performance with varying parameters. **a** Avg. no of items per transactions. **b** No of items

performance completely worst. In this case, MHUI-TID and our algorithm performance is somewhat better than the THUI-Mine algorithm. This is achieved due to patterns prefix sharing in used tree structure. From Fig. 5, we can observe that the parameter, number of distinct items, will not make any significant effect on all algorithms.

6 Conclusions

In this paper, we have proposed an algorithm based on sliding window called, RHUIM_{SW} for mining recent high utility itemsets over data streams. We have build PTable and HTable including the RHUI_{SW}-Tree to keep the information captured from sliding window and to make incremental update and tracing process faster. We have compared the performance against MHUI-TID and THUI-Mine. Experimental results shown that our approach is achieved good performance over THUI-Mine and MHUI-TID in terms of execution time and memory consumption.

References

1. Shen, Y.D., Zhang, Z., Yang, Q.: Objective-oriented utility-based association mining. In: Proceedings of 2002 IEEE International Conference on Data Mining, 2002, IEEE. ICDM 2002, pp. 426–433 (2002).
2. Yao, H., Hamilton, H.J., Butz, C.J.: A foundational approach to mining itemset utilities from databases. In: The 4th SIAM International Conference on Data Mining, pp. 482–486 (2004)
3. Yao, H., Hamilton, H.J.: Mining itemset utilities from transaction databases. Data Knowl. Eng. **59**(3), 603–626 (2006)
4. Agrawal, R., Srikant, R., et al.: Fast algorithms for mining association rules. In: Proceedings of 20th International Conference on Very Large Data Bases, VLDB, vol. 1215, pp. 487–499 (1994)
5. Liu, Y., Liao, W.K., Choudhary, A.: A two-phase algorithm for fast discovery of high utility itemsets. In: Ho, T., Cheung, D., Liu, H. (eds.) Advances in Knowledge Discovery and Data Mining, pp. 689–695. Springer, New York (2005)
6. Liu, Y., Liao, W.K., Choudhary, A.: A fast high utility itemsets mining algorithm. In: Proceedings of the 1st International Workshop on Utility-Based Data Mining, ACM, pp. 90–99 (2005)
7. Jiang, N., Gruenwald, L.: Research issues in data stream association rule mining. ACM Sigmod Rec. **35**(1), 14–19 (2006)
8. Cheung, D.W., Han, J., Ng, V.T., Wong, C.: Maintenance of discovered association rules in large databases: an incremental updating technique. In: Proceedings of the Twelfth International Conference on Data Engineering, IEEE, 1996, pp. 106–114 (1996)
9. Deypir, M., Sadreddini, M.H., Hashemi, S.: Towards a variable size sliding window model for frequent itemset mining over data streams. Comput. Ind. Eng. **63**(1), 161–172 (2012)
10. Chi, Y., Wang, H., Yu, P.S., Muntz, R.R.: Moment: maintaining closed frequent itemsets over a stream sliding window. In: Fourth IEEE International Conference on Data Mining, 2004, IEEE. ICDM'04, pp. 59–66 (2004)
11. Golab, L., Ozsu, M.T.: Issues in data stream management. ACM Sigmod Rec. **32**(2), 5–14 (2003)
12. Gaber, M.M., Zaslavsky, A., Krishnaswamy, S.: Mining data streams: a review. ACM Sigmod Rec **34**(2), 18–26 (2005)
13. Lee, C.H., Lin, C.R., Chen, M.S.: Sliding-window filtering: an efficient algorithm for incremental mining. In: Proceedings of the Tenth International Conference on Information and Knowledge Management, ACM, pp. 263–270 (2001)
14. Chan, R., Yang, Q., Shen, Y.D.: Mining high utility itemsets. In: Third IEEE International Conference on Data Mining, 2003, IEEE. ICDM 2003, pp. 19–26 (2003)
15. Chu, C.J., Tseng, V.S., Liang, T.: An efficient algorithm for mining temporal high utility itemsets from data streams. J. Syst. Softw. **81**(7), 1105–1117 (2008)
16. Li, H.F., Huang, H.Y., Chen, Y.C., Liu, Y.J., Lee, S.Y.: Fast and memory efficient mining of high utility itemsets in data streams. In: Eighth IEEE International Conference on Data Mining, 2008, IEEE. ICDM'08, pp. 881–886 (2008)