Online Mining of Temporal Maximal Utility Itemsets from Data Streams

Bai-En Shie
Dept. of Computer Science and Information Engineering
National Cheng Kung University, Taiwan, ROC
brian0326@idb.csie.ncku.edu.tw

Vincent S. Tseng
Dept. of Computer Science and Information Engineering
National Cheng Kung University, Taiwan, ROC
tsengsm@mail.ncku.edu.tw

Philip S. Yu
Dept. of Computer Science
University of Illinois at Chicago, Chicago, Illinois, USA
psyu@cs.uic.edu

ABSTRACT
Data stream mining has become an emerging research topic in the data mining field, and finding frequent itemsets is an important task in data stream mining with wide applications. Recently, utility mining is receiving extensive attentions with two issues re-considered: First, the utility (e.g., profit) of each item may be different in real applications; second, the frequent itemsets might not produce the highest utility. In this paper, we propose a novel algorithm named *GUIDE* (***G****eneration of temporal maximal* ***U****tility* ***I****temsets from* ***D****ata str****E****ams*) which can find temporal maximal utility itemsets from data streams. A novel data structure, namely, *TMUI-tree* (***T****emporal* ***M****aximal* ***U****tility* ***I****temset tree*), is also proposed for efficiently capturing the utility of each itemset with one-time scanning. The main contributions of this paper are as follows: 1) GUIDE is the first one-pass utility-based algorithm for mining temporal maximal utility itemsets from data streams, and 2) TMUI-tree is efficient and easy to maintain. The experimental results show that our approach outperforms other existing utility mining algorithms like Two-Phase algorithm under the data stream environments.

Categories and Subject Descriptors
H.2.8 [**Database Management**]: Database Applications - Data mining

General Terms
Algorithms

Keywords
Data stream mining, utility mining, maximal itemsets, temporal high utility itemsets

1. INTRODUCTION
Data stream analysis is an emerging topic extensively studied in recent decade. A data stream is a continuously ordered sequence of transactions that arrives sequentially in real-time manner. There are many applications of data stream mining, such as knowledge discovery from online e-business or transaction flows, analysis of network flows, monitoring of sensor data, and so on. Different from traditional databases, data streams have some special properties, namely continuous, unbounded, high speed and time-varying data distribution. Therefore, some limitations are posed in data stream mining as follows [9]. First, since the infinite transactions could not be stored, multi-scan algorithms are no more allowed. Second, in order to capture the information of the high speed data streams, the algorithms must be as fast as possible. Otherwise, the accuracy of the mining results will be reduced. Third, the data distribution of data streams should be kept to avoid concept drifting problem. Fourth, incremental processes are needed for mining data streams in order to make processing with the old data as little as possible. Therefore, efficient one-pass methods and compact data structures for characterizing the data stream are needed.

In recent years, utility mining emerges as a new research issue, which is to find the itemsets with high utilities, i.e., the itemsets whose utility values are greater than or equal to the user-specified minimum utility threshold. There exist rich applications of utility mining, such as business promotion, webpage organization and catalog design. Unlike traditional association rule mining, utility mining can find profitable itemsets which may not appear frequently in databases. By the advantages mentioned above, we can see that it is important to push utility mining into data stream mining, such as for the streaming data of the chain hypermarkets.

In view of this, we aim at finding maximal utility itemsets from data streams with the purpose of reducing the number of patterns in this paper. This is motivated by the observation that there exists no study that explores advanced utility itemset patterns like maximal utility itemsets in data streams. We proposed a novel algorithm called GUIDE (***G****eneration of temporal maximal* ***U****tility* ***I****temsets from* ***D****ata str****E****ams*) for finding maximal utility itemsets from the landmark time to the current time in data streams. The basic idea of GUIDE is to effectively pick up the essential information about the utilities of the appeared itemsets and store them into a compact data structure, namely TMUI-tree (***T****emporal* ***M****aximal* ***U****tility* ***I****temset tree*). The novel contributions of GUIDE are depicted as follows. First, to the best of our knowledge, it is the first one-pass algorithm for mining maximal high utility itemsets from data streams. Second, the proposed data structure, namely TMUI-tree, is easy to maintain for storing essential information in the mining processes and it facilitates GUIDE for finding temporal maximal utility itemsets efficiently. These nice features meet the requirements of data stream mining as mentioned previously. Through experimental evaluation, GUIDE is shown to outperform other relevant algorithms like Two-Phase [14] substantially since the nice feature of GUIDE that there is no need to generate candidate itemsets during the mining processes.

This paper is organized as follows. In section 2, we address some related researches. In section 3, we give some definitions and introduce the proposed method. In section 4, the performance of the proposed method is evaluated in the experiments. The conclusions are given in section 5.

2. RELATED WORKS

A number of researches have been proposed on mining data streams in the past several years. In [4, 5, 9], the issues, limitations and applications of data stream mining were discussed and overviewed in details. Further, many algorithms for mining different patterns from data streams were proposed such as Moment [3] and CFI-Stream [8] for mining closed frequent itemsets, estDec+ [10] and DSM-MFI [12] for mining maximal frequent itemsets, a time-sensitive model [13] and a two phase mining mechanism [17] for mining frequent itemsets. However, none of the above researches showed the expansibility and flexibility for the utility mining. THUI-Mine [16] was the first algorithm for mining high utility itemsets from data streams. It extends the Two-Phase algorithm [14] for data stream mining. Recently, Li et al. [11] proposed two algorithms, named MHUI-BIT and MHUI-TID, for mining high utility itemsets from data streams by a three-phase method. However, the methods in the above two researches will still scan the data in the sliding windows more than once which could not meet the limitations of data stream mining completely.

In these algorithms, three models, i.e., landmark, damped and sliding window, are used for capturing the data in data streams in different ways [9, 13]. Landmark model stores the whole data from the specific time point called landmark and finds patterns within the data [12]. In damped model, the whole data is also stored from the landmark time point. However, a weight is given for decreasing the importance of the out-of-date data [10]. In sliding window model, a window which slides with time is kept to capture the data within a fixed time or a fixed number of transactions. When the mining process is requested, just the data kept in the window is used [3, 8, 11, 13, 16, 17]. The three models are applicable to different situations according to users' demands.

In contrast to stream data mining, extensive researches on mining traditional databases have been done with rich kinds of patterns discovered. Apriori algorithm [1] was the pioneer for efficiently mining association rules from large databases. The tree-based association rule mining algorithms such as FP-growth [7] were afterward proposed. FP-growth improved the efficiency of mining association rules than Apriori substantially since it does not have to generate candidate itemsets and it scans database just twice. Many more algorithms for mining different patterns were proposed after Apriori, such as GenMax [6] for mining maximal frequent itemsets by a backtrack search tree, and CLOSET [15] for mining closed frequent itemsets by applying a compressed tree structure. Moreover, some new methods were proposed for mining high utility itemsets like [2] and [14]. Liu et al [14] proposed the Two-Phase algorithm to overcome the main problem of utility mining, i.e., the downward closure property is no longer suitable in utility mining, by applying the transaction-weighted downward closure property. The Two-Phase algorithm was shown to effectively reduce the number of the candidate itemsets and simplify the computations. However, Two-Phase is a multi-pass algorithm and it can not fit the limitation, i.e., one-pass, of data stream mining.

3. PROPOSED METHOD

In this section, we introduce a novel algorithm for mining temporal maximal utility itemsets from data streams. We define some terminologies used in this paper at first and then introduce the proposed algorithm, GUIDE. The main idea of GUIDE is to use a compact summary data structure, named TMUI-tree, for storing the utilities and the potential TMUIs in a new transaction. Thereafter, the actual TMUIs are found and generated in the data structure simultaneously.

3.1 Preliminary Concepts

In this section, we further extend the definitions of utility mining in [14] for the purpose of utility mining in data stream.

- $I = \{i_1, i_2, ..., i_m\}$ is a set of literals, called items. An itemset is a subset of I. Every item in the data stream has its own unit prize. The unit prize of the item x is denoted by $pr(x)$.
- A data stream DS is denoted by $\{(Tid_1, t_1), (Tid_2, t_2), ..., (Tid_n, t_n), ...\}$, where Tid_k ($k \geqq 1$) is a transaction and t_k ($k \geqq 1$) is the time when Tid_k appeared in DS.
- A transaction Tid_k is composed of a set of items and their purchased numbers, which is denoted by $\{(x_1, q_1), (x_2, q_2), ..., (x_p, q_p)\}$, where x_r ($1 \leqq r \leqq p$, $x_r \in I$) is the purchased item and q_r ($1 \leqq r \leqq p$) is the purchased number of x_r in Tid_k.
- The utility of the item x in Tid_k, which is denoted as $trans\_u_k(x)$, is defined as $pr(x)*q_r$. The utility of an itemset X in Tid_k, $trans\_u_k(X)$, i.e., the summarization of the items which are belonged to X, is defined as $\sum_{x \in X} trans\_u_k(x)$.
- The utility of the transaction Tid_k, denoted as $trans\_u_k(Tid_k)$, is defined as $\sum_{x \in Tid_k} trans\_u_k(x)$. The utility of an itemset X in DS, $u(X)$, is defined as $\sum_{k \in DS \wedge X \subseteq k} \sum_{x \in X} trans\_u_k(x)$.
- The total utility of a data stream, denoted as $totalU$, is defined as the summation of the utility of all transactions in the data stream which was accumulated progressively since the landmark was set.

By the above definitions, we define *temporal utility itemset*, denoted as *TUI*, as the itemset whose utility is greater than or equal to a user-specified minimum utility threshold s ($0 \leqq s \leqq 1$), where s is the percentage of the utility over total utility in DS. An itemset X is a TUI if its utility $u(X) \geqq totalU*s$. Moreover, by the basic definition of maximal frequent itemset in [6], we define *temporal maximal utility itemset*, denoted as *TMUI*, as the TUI which is not a subset of any other TUI. We call the maximal utility itemsets found from data streams temporal maximal utility itemsets because every pattern is evolving by time in data streams.

Problem Statement. Given a data stream, a pre-defined utility table and a user-specified minimum utility threshold, the problem of mining temporal maximal high utility itemsets using the landmark model is to find the set of TMUIs from the landmark timestamp over the entire history of the data stream.

3.2 Definitions of Terminologies

We introduce some basic definitions of the terminologies in the following paragraphs.

Definition 1. The components of TMUI-tree. A TMUI-tree is composed of nodes and links. An itemset will be stored as a node in TMUI-tree. The node of TMUI-tree, denoted as *<id: utility:*

time>, is a triple consisting of the identity of the itemset, the utility of the itemset, and the time when the itemset was added as a node into the TMUI-tree. Each node has two kinds of links, one is parent-link which points to its parent node and another is children-link which points to its children node.

In the GUIDE algorithm, when a new transaction arrives, we must decompose the transaction into several projections and then store them into TMUI-tree. The definition of the method, i.e. transaction-projection, is as follows.

Definition 2. Transaction-projection. Without loss of generality, in this paper, we assume that the itemsets in all transactions are sorted in a certain fixed order, e.g. in alphabetical order. When a transaction $\{(i_1, q_1)\ (i_2, q_2)\ ...\ (i_n, q_n)\}$ arrives, we generate the projections of this transaction. Instead of generating all sub-itemsets of this transaction, we proposed the transaction-projection which is described as follows. First, the transaction is projected into the postfixes by pruning the first item in the transaction in turn, i.e., $\{(i_1, q_1)\ (i_2, q_2)\ ...\ (i_n, q_n)\}$, $\{(i_2, q_2)\ ...\ (i_n, q_n)\}$, ..., and $\{(i_n, q_n)\}$. Then, for each postfix, it is projected into the prefixes by pruning the last item in turn. Taking $\{(i_2, q_2)\ (i_3, q_3)\ ...\ (i_n, q_n)\}$ for example, we get $\{(i_2, q_2)\ (i_3, q_3)\ ...\ (i_n, q_n)\}$, $\{(i_2, q_2)\ (i_3, q_3)\ ...\ (i_{n-1}, q_{n-1})\}$, ..., and $\{(i_2, q_2)\}$.

When a new TUI was generated from TMUI-tree, it will be tested whether it is maximal or not. If it is maximal, it will be stored into a temp list which is called TMUI-list. When a user queries GUIDE for all TMUIs from the landmark time to the present, we will output the list. The TMUI-list is defined as follows.

Definition 3. TMUI-list. A TMUI-list is a set of all TMUIs from the landmark time to the present. Each TMUI in the TMUI-list is represented by a pair, denoted as <*id: utility*>, where id stands for the identity of the TMUI, utility means the summation of the utility value of this TMUI from the landmark time to the present in the data stream.

3.3 Proposed Algorithm: GUIDE

We introduce the GUIDE algorithm which we proposed for finding TMUIs from data stream in detail as follows. The GUIDE algorithm is shown in Figure 1. When a new transaction arrives into the data stream, we perform transaction-projection on this transaction and get its projections. Then, each item in the projections is checked whether there is a node in TMUI-tree for the item. If the node did not exist, it is created. Otherwise, its utility is accumulated into the node. At the same time, the utility of each modified node *X* is examined to see whether it is greater than or equal to the minimum utility threshold. If a new TUI was generated, i.e. $u(X) \geqq totalU^*s$, it is also examined whether it is maximum. If it is not a subset of any other TMUI in TMUI-list, it is added into the TMUI-list. At last, an additional check is performed. Some TMUIs in the TMUI-list may be invalid since they would be expired as the time goes by. In this checking process, each TMUI in the TMUI-list is checked whether its utility is above than or equal to the new threshold, i.e. $totalU_{now}{}^*s$. Because $totalU_{now}{}^*s$ will be increased when a new transaction arrives, the TMUIs in TMUI-list whose utility become smaller than the new threshold must be pruned. If the utility of a TMUI *Y* is above than or equal to the threshold, i.e., $u(Y) \geqq totalU_{now}{}^*s$, it is kept; otherwise, it is pruned.

To illustrate by an example, assume that there is a data stream *DS* and a utility table for the items in *DS* as shown in Figure 2. In Figure 2(a), the first column "*Tid, Time*" means the time when the transaction comes into *DS*. We set the minimum utility threshold *s* as 20% and the landmark time is at time 0. According to the transaction time, the first transaction Tid_1, that is $\{(C, 12)\ (E, 2)\}$, is arriving into *DS* at time 2. After transaction-projection, we have three projections, namely $\{C\}$, $\{E\}$ and $\{CE\}$. The utility of these projections are 12, 10 and 22, respectively. Because TMUI-tree is null now, we add all nodes with regard to all these projections. The node <*C: 12: 2*> is first added as a child node of the root of the TMUI-tree. Then the node <*E: 22: 2*> is added as a child of the node <*C: 12: 2*>. It stands for the itemset $\{CE\}$. Finally, the node <*E: 10: 2*> is added as another child of the root. The TMUI-tree now is shown in Figure 3(a). During adding the first transaction into the TMUI-tree, the modified nodes are checked whether they are new TMUIs. At present, the *totalU* of *DS* is 22 and the threshold $totalU^*s$ is 4.4. Therefore, the itemsets <*C: 12*>, <*CE: 22*> and <*E: 10*> are all new TUIs which are above the threshold. After examined the maximal property of the three TUIs, only <*CE: 22*> is a TMUI among them and it is added into the TMUI-list at time 2. Figure 3 shows the TMUI-tree when all transactions in *DS* were analyzed by GUIDE. The gray nodes stand for the nodes which were modified or created at that time.

Procedure: GUIDE
Input: A data stream *DS*, a pre-defined utility table and a minimum utility threshold *s*
Output: A list of currently TMUIs
1. **while** a new transaction Tid_k arrives into *DS* **do**
2. $totalU = totalU + trans\_u_k(Tid_k)$
3. perform transaction-projection on Tid_k and get its projections $proj_k$
4. **for** each item $i \in p,\ p \in proj_k$ **do**
5. $tmp\_util = tmp\_util + trans\_u_k(i)$ //a temp utility value of *i*
6. trace TMUI-tree to the node of *i* by the identity of *p*
7. **if** the node do not exist **do**
8. create a new node <*i*'s id: *tmp_util*: t_k> in TMUI-tree
9. **else** modify the node's utility by adding *tmp_util*
10. **if** the node's utility $\geq totalU^*s$ **do**
11. create a new TUI *X* <*node's id*: *node's utility*>
12. **if** X is maximum **do** add *X* into the TMUI-list
13. **end while**
14. **for** each TMUI in TMUI-list **do**
15. **if** its utility is less than $totalU^*s$ **do** prune the TMUI

Figure 1. The procedure of GUIDE

Tid, Time	Transaction
(1,2)	(C,12)(E,2)
(2,3)	(B,6)(D,1)(E,1)
(3,6)	(A,1)(B,4)(E,1)
(4,8)	(A,1)(C,1)(D,15)(E,1)
(5,9)	(A,1)(B,1)(C,27)

(a)

Item	Profit
A	3
B	10
C	1
D	6
E	5

(b)

Figure 2. (a) A data stream *DS* (b) A utility table

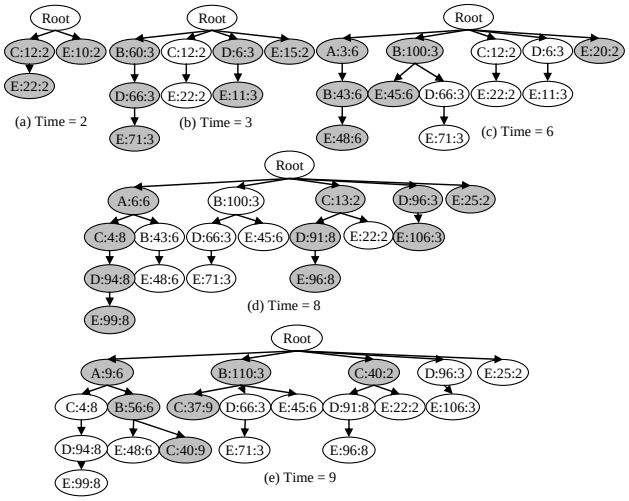


Figure 3. An example of the construction of TMUI-tree

3.4 The Pruning Method of TMUI-tree

Since the data streams are infinite, we can not store all data into TMUI-tree for a long duration time. Therefore, we propose a pruning method for TMUI-tree to prevent the memory exhaustion. Before describing the pruning method, an additional definition for a notation *PT(X)* is depicted as follows.

- *PT(X)*, the ratio of pass time of the itemset *X* appeared in the data stream *DS*, is defined as $(t_{now}-t_X)/(t_{now}-t_{LM})$, where t_{now} is current time of *DS*, t_{LM} is the landmark time of *DS*, and t_X is the time when the itemset *X* was firstly appeared in *DS*. Without loss of generality, we also defined the pass time of the landmark time and the current time, i.e., *PT(LM)* and *PT(now)*, as 1 and 0.

We describe the proposed pruning method of the TMUI-tree in detail as follows. Assume that the minimum utility threshold is *s*. Since there is no downward closure in utility mining, we must check every node in TMUI-tree without skipping any node. When the pruning process is requested, each node in TMUI-tree is checked orderly to decide whether it will be pruned. The pruning threshold for a node *x* is *s*totalU*PT(x)*. If the utility of *x* was less than the pruning threshold, *x* is pruned. The threshold is determined by the time when the node was added into TMUI-tree. In other words, if a node was added into TMUI-tree lately, the pruning threshold of this node is small; otherwise the threshold is large. Thus, the nodes added early with less utility are pruned; on the other hand, the nodes added lately with larger utility are kept.

4. EXPERIMENTAL EVALUATIONS

To evaluate the performance of GUIDE, we have conducted several experiments by using synthetic datasets generated from the IBM synthetic data generator as in [1]. For other parameters related to utility mining, we use the settings the same to [14]. The description of parameters and the default settings are shown in Table 1. The simulation was implemented in Java and conducted in a machine with 3.0GHz CPU and 1GB memory. To the best of our knowledge, although there are several past researches which dealt with utility mining or data stream mining, there is still no single-pass algorithm which can find maximal temporal utility itemsets from data streams. Hence, we compare our method with Two-Phase, which is the state-of-the-art algorithm for mining high utility itemsets in traditional databases. In the following paragraphs, we conduct four parts of experiments to evaluate the performance of the GUIDE algorithm.

Table 1. Parameter settings

Parameter Descriptions	Default
D: Number of transactions in data stream	10k, 100k
T: Average size of the transactions	10, 20
I: Average size of the potentially high utility itemsets	6
L: Number of potentially high utility itemsets	500
N: Number of distinct items	1000
q: Number of purchased items in transactions	1~5
pr: Unit price of each item	1~1000

Experiment I. Effects of varying minimum utility thresholds. In the first part of the experiments, we compare the execution time of GUIDE and Two-Phase. As shown in Figure 4(a), we can see the results of the two algorithms in the dataset T10I6D10k. Although the execution time of Two-Phase decreases obviously with the minimum utility threshold increased, GUIDE still performs better than Two-Phase when minimum utility threshold is between 0.2% and 1%. It can be also observed that GUIDE is more stable than Two-Phase because the execution time of GUIDE does not increase as sharply as Two-Phase when the minimum utility threshold is decreased.

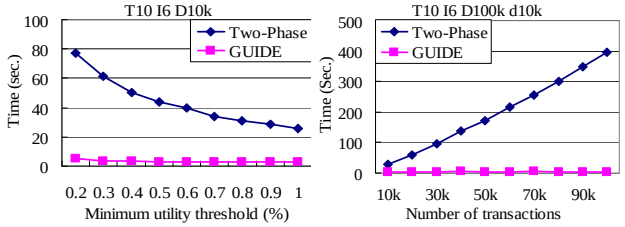
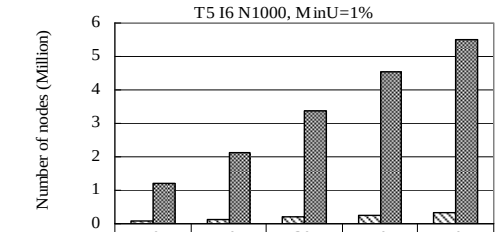


Figure 4. Execution time
(a) under varied minimum utility thresholds
(b) under varied number of transactions

Experiment II. Performance comparisons under stream environments. In the second part of the experiments, we increase the number of transactions by inserting new transactions to simulate the streaming data mining environment, i.e., restricting the processes to one-scan, to see the differences of performance between GUIDE and Two-Phase. In this experiment, we set 10,000 transactions as a batch and input a batch at a time, i.e., the size of the incremental transactions is once 10,000. Because Two-Phase is a multi-pass algorithm which is not designed for mining data streams, it re-scans the whole data when a new batch arrives. However, GUIDE just processes the new inserted 10,000 transactions without re-scanning other old data. The experimental results are shown in Figure 4(b). We can see that the execution time of Two-Phase increases regularly with the number of total transactions since it needs to re-scan the whole data when a new batch arrives. On the other hand, GUIDE just deals with the inserted transactions when a new batch arrives. Its execution time remains steady at each batch. It shows that one-pass algorithm is necessary for mining data streams. Multi-pass algorithms which must re-scan the whole database can not perform efficiently, that is, they cannot fit the limitations of streaming data environments.

Experiment III. Compactness of TMUI-tree. In this part, we show the compactness of TMUI-tree. The results are shown in

Figure 5. In this figure, *All_nodes* stands for the number of nodes in the trees whose projections are all combinations of the transactions and, on the contrary, *TMUI-tree* stands for the number of nodes in the trees whose projections are generated by the transaction-projection of GUIDE. In this figure, we can see that the number of nodes of *TMUI-tree* is linearly increased with the dataset size increasing. Meanwhile, the number of nodes in *All_nodes* keeps about 15 times of those in *TMUI-tree*. By this experiment, we can see that the TMUI-tree is a compact and scalable tree structure.



Dataset size(k)	10	20	30	40	50
TMUI-tree	77815	143402	211919	268370	323467
All_nodes	1189200	2129499	3369939	4526694	5483337

Figure 5. Number of nodes in TMUI-tree under varied number of transactions

Experiment IV. Completeness of GUIDE. Since GUIDE is an approximate algorithm, it may loss some patterns. In the following experiments, the completeness of GUIDE is examined. Without loss of generality, we first define the measurement named *recall of patterns*, abbreviated as *ROP*, as the percentage of the patterns captured by GUIDE. The formula of ROP is shown below. ROP reflects the completeness of found patterns.

$$ROP(\%) = \frac{\text{number of patterns found by GUIDE}}{\text{number of all patterns should be found}} \times 100\%$$

We perform the experiment by varying the size of datasets. Table 2 is the experiment results of GUIDE under the dataset T5I6N1000. The minimum utility threshold is set as 1% and D is set from 10,000 to 50,000. In this table, the second column *#patterns* is the number of all TMUIs which should be found from dataset and, on the contrary, the third column *#GUIDE* is the number of TMUIs which is actually found by GUIDE. It can be seen from this that as the size of dataset increases, the ROP of GUIDE is gradually decreased. It shows that GUIDE is more suitable for small datasets. Moreover, in the experiment, the ROPs of the different datasets are almost above 97%. This means that GUIDE has high completeness of patterns for mining TMUIs in data streams.

Table 2. The recall of patterns under different dataset sizes

D(k)	#Patterns	#GUIDE	ROP (%)
10	100	99	99
20	104	103	99.04
30	95	94	98.95
40	115	113	98.26
50	87	85	97.70

By the above experiments, we can show that although GUIDE misses a few patterns, it spends much lower execution time than Two-Phase. Further, TMUI-tree is proven for its completeness and compactness. Thus GUIDE is an efficient and effective algorithm for mining TMUIs in data streams.

5. CONCLUSIONS

In this paper, we proposed a novel algorithm, namely GUIDE, for efficiently mining temporal maximal utility itemsets from the landmark time to the present in data streams. We also proposed a new data structure, namely TMUI-tree, for storing information in the processes of mining utility patterns from data streams. The main contributions of GUIDE and TMUI-tree are that GUIDE is the first one-pass algorithm for mining maximal utility itemsets in data streams and TMUI-tree is easy to maintain and it can help GUIDE finding TMUIs efficiently. The experimental results on different datasets showed that GUIDE is substantially efficient than the state-of-the-art utility mining algorithm, i.e. the Two-Phase algorithm, under different conditions. Further, the results showed that GUIDE has high compactness and high completeness of patterns. For future works, we will apply and extend the GUIDE algorithm to the sliding window model and time-fading model for data stream mining. Advanced evaluations and experimental evaluations on GUIDE will also be conducted.

6. ACKNOWLEDGMENTS

This research was supported by National Science Council, Taiwan, R.O.C. under grant no. NSC 96-2221-E-006-143-MY3 and NSC 98-2631-H-006 -001.

7. REFERENCES

[1] Agrawal, R. and Srikant, R. Fast algorithms for mining association rules. In *Proc. of the 20th Int'l Conf. on Very Large Data Bases*, 1994, 487-499.

[2] Chan, R., Yang, Q. and Shen, Y. Mining high utility itemsets. In *Proc. of Third IEEE Int'l Conf. on Data Mining*, Nov., 2003, 19-26.

[3] Chi, Y., Wang, H., Yu, P. S. and Muntz, R. R. Moment: maintaining closed frequent itemsets over a stream sliding window. In *Proc. of the IEEE Int'l Conf. on Data Mining*, 2004.

[4] Gaber, M. M., Zaslavsky, A. B. and Krishnaswamy, S. Mining data streams: a review. *ACM SIGMOD Record*, Vol.34, No.2, 2005, 18-26.

[5] Golab, L. and Ozsu, M. T. Issues in data stream management. In *ACM SIGMOD Record*, Vol. 32, No. 2, 2003.

[6] Gouda, K. and Zaki, M. J. Efficiently mining maximal frequent itemsets. In *Proc. of the IEEE Int'l Conf. on Data Mining*, 2001.

[7] Han, J., Pei, J., and Yin, Y. Mining frequent patterns without candidate generation. In *Proc. of the ACM-SIGMOD Int'l Conf. on Management of Data*, 2000, 1-12.

[8] Jiang, N. and Gruenwald, L. CFI-Stream: mining closed frequent itemsets in data streams. In *Proc. of the Utility-Based Data Mining Workshop, ACM KDD*, 2006.

[9] Jiang, N. and Gruenwald, L. Research issues in data stream association rule mining. In *ACM SIGMOD Record*, Vol. 35, No. 1, 2006.

[10] Lee, D. and Lee, W. Finding maximal frequent itemsets over online data streams adaptively. In *Proc. of 5th IEEE Int'l Conf. on Data Mining*, 2005.

[11] 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 Proc. of *the 8th IEEE Int'l Conf. on Data Mining*, 2008, 881-886.

[12] Li, H.-F., Lee, S.-Y., and Shan, M.-K. Online mining (recently) maximal frequent itemsets over data streams. In *Proc. of the 15th IEEE Int'l Workshop on Research Issues on Data Engineering*, 2005.

[13] Lin, C. H., Chiu, D. Y., Wu, Y. H. and Chen, A. L. P. Mining frequent itemsets from data streams with a time-sensitive sliding window. In *Proc. of SDM*, 2005.

[14] Liu, Y., Liao, W. and Choudhary, A. A fast high utility itemsets mining algorithm. In *Proc. of the Utility-Based Data Mining Workshop*, 2005.

[15] Pei, J., Han, J., and Mao, R. CLOSET: An efficient algorithm for mining frequent closed itemsets. In *Proc. of the ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2000, 11-20.

[16] Tseng, V. S., Chu, C. J. and Liang, T. Efficient mining of temporal high utility itemsets from data streams. In *ACM KDD Workshop on Utility-Based Data Mining Workshop*, Philadelphia, USA, 2006.

[17] Tanbeer, S. K., Ahmed, C. F., Jeong, B.-S. and Lee,Y.-K. Efficient frequent pattern mining over data streams. In *Proc. of the ACM 17th Conference on Information and Knowledge Management*, 2008.