Efficient Algorithms for Mining Maximal High Utility Itemsets from Data Streams with Different Models

Bai-En Shie[1], Philip S. Yu[2] and Vincent S. Tseng [1+]

[1] Department of Computer Science and Information Engineering,
National Cheng Kung University, Tainan 701, Taiwan, ROC
[2] Department of Computer Science, University of Illinois at Chicago,
Chicago, Illinois, USA
brianshie@gmail.com, psyu@cs.uic.edu, tsengsm@mail.ncku.edu.tw

Abstract. Data stream mining is an emerging research topic in the data mining field. Finding frequent itemsets is one of the most important tasks in data stream mining with wide applications like online e-business and web click-stream analysis. However, two main problems existed in relevant studies: 1) The utilities (e.g., importance or profits) of items are not considered. Actual utilities of patterns can not be reflected in frequent itemsets. 2) Existing utility mining methods produce too many patterns and this makes it difficult for the users to filter useful patterns among the huge set of patterns. In view of this, in this paper we propose a novel framework, named *GUIDE* (*Generation of maximal high Utility Itemsets from Data strEams*), to find maximal high utility itemsets from data streams with different models, i.e., landmark, sliding window and time fading models. The proposed structure, named *MUI-Tree* (*Maximal high Utility Itemset Tree*), maintains essential information for the mining processes and the proposed strategies further facilitates the performance of GUIDE. Main contributions of this paper are as follows: 1) To the best of our knowledge, this is the first work on mining the compact form of high utility patterns from data streams; 2) GUIDE is an effective one-pass framework which meets the requirements of data stream mining; 3) GUIDE generates novel patterns which are not only high utility but also maximal, which provide compact and insightful hidden information in the data streams. Experimental results show that our approach outperforms the state-of-the-art algorithms under various conditions in data stream environments on different models.

Keywords: high utility itemset, maximal pattern, utility mining, data stream mining

1 Introduction

A data stream is composed of continuously ordered data that arrive sequentially in real-time manner. Data stream analysis is an emerging issue extensively studied in recent decade [3, 5~10, 13~20, 22, 24, 26, 27, 30, 31]. Data stream mining has many

applications, such as knowledge discovery from online e-business or transaction flows, network flow analysis, monitoring of sensor data, and web log and click-stream mining. For different applications, there are three models commonly used in data streams: *landmark*, *sliding window* and *time fading models* [6, 14, 22]. Landmark model stores the whole data from a specific time point called *landmark* and finds patterns within the data [24, 26, 30]. Sliding window model uses a *fix-sized window* which slides with time to keep the data within fixed time or a fixed number of transactions. In the mining process, only the data kept in the window are considered [3, 5, 7, 9, 13, 17, 19, 20, 22, 27, 31]. Time fading model also captures data from the landmark time to the present; however, in this model, a *time decay function* is given to decrease the importance of out-of-date data [16, 18]. Different from traditional databases, data streams have some special properties: continuous, unbounded, coming with high speed and time-varying data distribution. Therefore, discovering knowledge from data streams poses some limitations as follows. First, since the infinite data cannot be stored, traditional multi-scan algorithms are no more allowed. Second, in order to capture the information of high speed data streams, the algorithm must be as fast as possible; otherwise, the accuracy of mining results will be decreased. Third, the data distribution within the data streams should be kept to avoid concept drifting problem. Fourth, it needs incremental processes to process the existing data as less as possible. Therefore, to discover useful information from data streams needs not only efficient one-pass algorithms but also effective data summary techniques [6, 8, 10, 14].

There are many researches addressing the topic of knowledge discovery from data streams [3, 5~10, 13~19, 22, 24, 26, 30, 31]. However, two main problems exist in these works: First, actual profits of items are not considered. In many applications, such as e-business, this factor is often one of the most important factors for the results. To conquer this problem, *utility mining* [2, 4, 21, 23, 28, 29] emerges as a new research issue for discovering the itemsets with high utilities, i.e., high profits. There are rich applications for utility mining, such as business promotion in hypermarkets or chain stores, webpage organization and website click streaming analysis, and some biomedical applications. Unlike traditional frequent pattern mining, utility mining finds profitable patterns which may not appear frequently in transaction databases. Therefore, we can realize that to push utility mining into the data stream is an essential topic, so that relevant applications can be facilitated.

The second problem is that the previous works produce a huge set of patterns. It needs to apply heavy and empirical post-processes on the results; otherwise the patterns are too many to be helpful for users. Some previous researches addressed the topic of mining compact patterns in static databases [11, 25]. *Maximal* and *closed* itemsets are two kinds of meaningful compact patterns. An itemset is called maximal if it is not a subset of any other itemsets [11]. An itemset is called closed if it has no superset with the same support count [25]. The two kinds of compact patterns can not only reduce the huge number of frequent patterns but also provide meaningful results. Nevertheless, although some researches addressed the topic of mining closed or maximal patterns from data streams [3, 5, 7, 11, 13, 16, 19, 24, 30, 31], none of them consider utilities of itemsets in the mining

process. On the other hand, although some researches addressed the topic of mining high utility patterns from data streams [20, 27], none of them provide the compact form for patterns. Thus users may need to filter useful patterns within a huge amount of patterns by performing extra processes.

In view of these, in this paper, we investigate the topic of finding *maximal high utility itemsets*, which are not only high utility but also maximal, from data streams. A novel framework called *GUIDE* (*Generation of maximal high Utility Itemsets from Data strEams*) is proposed for finding maximal high utility itemsets from data streams. Based on the proposed framework, three algorithms, namely $GUIDE_{LM}$, $GUIDE_{SW}$ and $GUIDE_{TF}$, are proposed for landmark, sliding window and time fading models, respectively. The basic idea of the proposed algorithms is to effectively pick up the essential information, i.e., the utilities of appeared itemsets, and store them into tree structures, namely *MUI-Trees* (***M****aximal high* ***U****tility* ***I****temset Trees*). To facilitate the mining process, two strategies are proposed for efficient tracing and pruning the MUI-Trees.

Major contributions of this work are depicted as follows. First, to the best of our knowledge, this is the first work for mining the compact form of high utility itemsets from data streams with different models. Second, GUIDE is an effective one-pass framework which fits to the limitations of data streams as mentioned previously. Third, MUI-Trees maintain essential information for the mining processes and the two proposed strategies further facilitate the performance of GUIDE. Fourth, GUIDE generates the novel patterns which are not only high utility but also maximal. The patterns provide compact and insightful hidden information in the data streams. Through experimental evaluation, the proposed algorithms are shown to substantially outperform the state-of-the-art algorithms for mining high utility itemsets from data streams [20, 27].

This paper is organized as follows. In Section 2, we discuss about relevant researches about this topic. In Section 3, definitions and problem statements are described. Section 4 introduces the proposed framework GUIDE and the proposed methods for mining maximal high utility itemsets from data streams with different models. In Section 5, the performance of GUIDE is evaluated and discussed. The conclusions are given in Section 6.

2 Related Work

Literature reviews about frequent pattern mining, utility mining and data stream mining are given in this section.

2.1 Frequent Pattern Mining

The researches on mining patterns from traditional transaction databases have been developed for many years and rich kinds of patterns are discovered for different purposes.

Frequent pattern mining [1, 12] is the most popular topic among them. Apriori algorithm [1] is the pioneer for efficiently mining frequent itemsets from large databases. Tree-based frequent itemset mining algorithms such as FP-Growth [12] are afterward proposed. FP-Growth improves the efficiency of mining frequent itemsets than Apriori substantially since it does not have to generate candidate itemsets and it scans database just twice. However, the main problem of the traditional frequent pattern mining is that it produces too many patterns to be effectively applied. Users may need to filter out useless patterns by themselves. However, to pick up meaningful compact patterns is a heavy and empirical work even the process is performed by experts. Thus the problem of mining compact form of patterns has been raised. Algorithms for mining compact patterns are proposed, such as GenMax [11] for mining maximal frequent itemsets by a backtrack search tree, and CLOSET [25] for mining closed frequent itemsets by applying a compressed tree structure. Both the two kinds of patterns provide compact forms of meaningful patterns and effectively reduce the number of patterns.

2.2 Utility Mining

In frequent pattern mining, unit profits and purchased quantities of the items are not considered. Thus the issue of utility mining [2, 4, 21, 23, 28, 29] is proposed for dealing with this problem. UMining algorithm [29] proposed by Yao et al. used an estimated method to prune the search space. Liu et al. proposed an algorithm called Two-Phase [23] which applies the transaction-weighted downward closure property to reduce the search space for finding high utility itemsets. It first discovers high transaction-weighted utilization itemsets (abbreviated as HTWUIs) in phase I and then checks their real utilities in phase II to find actual high utility itemsets by additional database scans. Although Two-Phase algorithm reduces the search space of utility mining, it still generates too many candidates. Thus, Li et al. [21] proposed an isolated items discarding strategy (abbreviated as IIDS) to reduce the number of candidates by pruning isolated items during the level-wise pattern generation processes. Although the above algorithms prune the search space of utility mining increasingly, they also utilize the apriori-based candidate-generation-and-testing framework which needs to scan database for multiple times.

On the other hand, Ahmed et al. [2] proposed a structure named IHUP-Tree for maintaining essential information about utility mining. It avoids scanning database for multiple times and generating candidates during the mining process. However, although IHUP-Tree achieves better performance than Two-Phase and IIDS, it still produces too many HTWUIs. In view of this, Tseng et al. proposed a novel algorithm named UP-Growth [28], which applies several pruning and counting strategies during the mining processes. By the proposed strategies, the estimated utilities are effectively decreased in UP-Trees during the mining processes and the number of HTWUIs is further reduced. Therefore, the performance of utility mining can be improved significantly.

2.3 Data Stream Mining

In contrast to discovering patterns from static databases, there are many studies proposed for data stream mining [3, 5~10, 13~20, 22, 24, 26, 27, 30, 31]. In [6, 8, 10, 14], the issues, limitations and applications of data stream mining are discussed and reviewed in detail. Many algorithms for mining various kinds of patterns from data streams are proposed such as Moment [7], CFI-Stream [13], IncMine [5], NewMoment [19], FP-CDS [24], and CloStream [30, 31] for mining closed frequent itemsets; estDec+ [16] for mining maximal frequent itemsets; FP-stream [9], time-sensitive model [22] and GraphMiner algorithms [3] for mining frequent closed graphs.

However, none of the above algorithms show the expansibility and flexibility for utility mining. THUI-Mine [27] is the first algorithm for mining high utility itemsets from data streams. It extends the framework of Two-Phase algorithm [23] to the data stream mining. Afterward, Li et al. [20] proposed two algorithms, named MHUI-BIT and MHUI-TID, for mining high utility itemsets from data streams by a three-phased method. MHUI-TID is shown to be the state-of-the-art algorithm for mining high utility itemsets from data streams. However, the above listed methods [20, 27] still need to spend more memory space for storing the original data within the window in main memory and then re-scan the data for several times. They could not meet the requirements of data stream mining completely. Thus, the methods can not be applied to landmark and time fading models since all data need to be accumulated from the landmark time to the present, in other words, all historical data should be kept.

In this paper, we combine the concepts of data stream mining, utility mining, and maximal pattern mining to discover maximal high utility patterns from data streams. To the best of our knowledge, this is the first work that finds the compact form of high utility itemsets for different models of data stream mining. By the proposed framework and methods, not only the mining performance can be enhanced but also the discovered patterns can be more compact and meaningful.

3 Preliminary and Definition

In this section, we extend the definitions of utility mining from the transaction databases to the data stream. We first define the notations for utility mining in streaming data environments and then address the problem statement of this work.

A *data stream DS* is composed of a continuous set of transactions denoted by $\{Tid_1, Tid_2, ..., Tid_n, ...\}$. A *transaction* Tid_k is denoted as $\{itemset_k, t_k\}$, where $itemset_k (k \geq 1)$ is the items appeared in the transaction Tid_k and t_k $(k \geq 1)$ is the time when $itemset_k$ appeared in *DS*. $Itemset_k$ 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 $(1 \leq r \leq p, x_r \in I)$ is the *item* and q_r $(1 \leq r \leq p)$ is the *purchased number* of x_r in Tid_k. $I = \{i_1, i_2, ..., i_m\}$ is the set of items. An *itemset* is a subset of *I*. Every item in the data stream has its own unit profit. The *unit*

profit of an item x, which is recorded in a *utility table*, is denoted by $p(x)$.

Definition 1. (Valid transaction.) For a transaction $Tid_k = \{itemset_k, t_k\}$, Tid_k is *valid* in the data stream *DS* if it is *captured* by following conditions:

1. For landmark and time fading models, assume the landmark time point is set as t_{lm}. Tid_k is valid if $t_{now} \geq t_k > t_{lm}$.
2. For sliding window model, assume window size is set as t_{sw}. Tid_k is valid if $t_{now} \geq t_k > t_{now} - t_{sw}$, where t_{now} is the current time of *DS*.

Definition 2. (Utilities of the elements in a data stream.) The utilities of items, itemsets and transactions in the data stream *DS* are defined as follows.

1. The utility of an item x_r in a transaction Tid_k is denoted as $u(x_r, Tid_k)$ and defined as $p(x_r) \times q_r$.
2. The utility of an itemset X in Tid_k is the summarization of the utilities of the items those are belonged to X in Tid_k. It is denoted and defined as $u(X, Tid_k) = \sum_{\forall x \in X} u(x, Tid_k)$.
3. The utility of X in *DS*, which is denoted as $u(X)$, is defined as $\sum_{\forall Tid_k \supseteq X \wedge Tid_k \in DS} u(X, Tid_k)$.
4. The transaction utility of Tid_k is denoted and defined as $u(Tid_k) = \sum_{\forall x \in Tid_k} u(x, Tid_k)$.
5. Total utility of *DS* is the summation of the utility of all valid transactions in *DS*. Assume the set of valid transactions in *DS* is *VT*. It is denoted and defined as $TotalU = \sum_{\forall Tid_k \in VT} u(Tid_k)$. By Definition 1, for landmark and time fading models, it is accumulated progressively since the landmark was set; for sliding window model, only the utilities of the transactions in the window are accumulated.

Definition 3. (High utility itemset and maximal high utility itemset.) By the above definitions, we define *high utility itemset*, abbreviated as *HUI*, as the itemset whose utility is larger than or equal to the total utility multiplies a user-specified minimum utility threshold *MinU*, where $0 \leq MinU \leq 1$. In other words, an itemset X is a HUI if $u(X) \geq TotalU \times MinU$. Moreover, by the definition of maximal frequent itemset in [11], we define *maximal high utility itemset*, abbreviated as *MaxHUI*, as a HUI which is not a subset of any other HUI.

A running example of the above definitions is shown as follows. Assume that there are a continuous data stream *DS* and a utility table as shown in Tables 1 and 2, respectively. In *DS*, the first transaction Tid_1 is appeared at time 2. The purchased items in Tid_1 are {C} and {E} with purchased numbers 12 and 2, respectively. The utility of item {C} in Tid_1, $u(\{C\}, Tid_1)$, is calculated as $1 \times 12 = 12$. The utility of itemset {CE} in Tid_1, i.e., $u(\{CE\}, Tid_1)$, is calculated as $1 \times 12 + 5 \times 2 = 22$. Besides, the utility of {CE} in *DS*, i.e., $u(\{CE\})$, is calculated as $u(\{CE\}, Tid_1) + u(\{CE\}, Tid_4) = 22 + 6 = 28$, since {CE} appears in Tid_1 and Tid_4. Moreover, the transaction utility of Tid_1 is calculated as $1 \times 12 + 5 \times 2 = 22$. *TotalU* of *DS* after time 9 is 280. If we set *MinU* to 5%, {CE} will be a HUI after time 9 because $u(\{CE\})$ is larger than $TotalU \times MinU$, that is, $28 > 280 \times 5\% = 14$. However,

{CE} cannot be a MaxHUI because there exists a HUI {ACDE} which is a superset of {CE}.

Problem Statement. Given a continuous data stream *DS*, a pre-defined utility table and a user-specified minimum utility threshold *MinU*, the problem of mining maximal high utility itemsets with different models is to find the set of MaxHUIs within the valid transactions for different models in data streams.

Table 1. A data stream DS.

Transaction	Itemset
(T_1, 2)	(C, 12) (E, 2)
(T_2, 4)	(B, 6) (D, 1) (E, 1)
(T_3, 6)	(A, 1) (B, 4) (E, 1)
(T_4, 8)	(A, 1) (C, 1) (D, 15) (E, 1)
(T_5, 9)	(A, 1) (B, 1) (C, 27)

Table 2. A utility table.

Item	Profit (per unit)
A	3
B	10
C	1
D	6
E	5

4 Proposed Framework: GUIDE

In this section, we introduce the proposed framework *GUIDE* (*Generation of maximal high Utility Itemsets from Data strEams*) for mining maximal high utility itemsets from data streams. The flowchart of GUIDE is shown in Figure 1. GUIDE mainly contains four steps: 1) Transaction-projection, 2) update the MUI-Tree, 3) pattern generation by tracing the MUI-Tree, and 4) MUI-Tree pruning. In the following paragraphs and subsections, we will introduce each step in turn in details.

Initially, the landmark time or the sliding window is set. Then the incoming transactions are loaded into memory and a process named *transaction-projection* is applied for producing the subsets of the transactions, called *projections*. The procedure of the transaction-projection is shown in Figure 2.

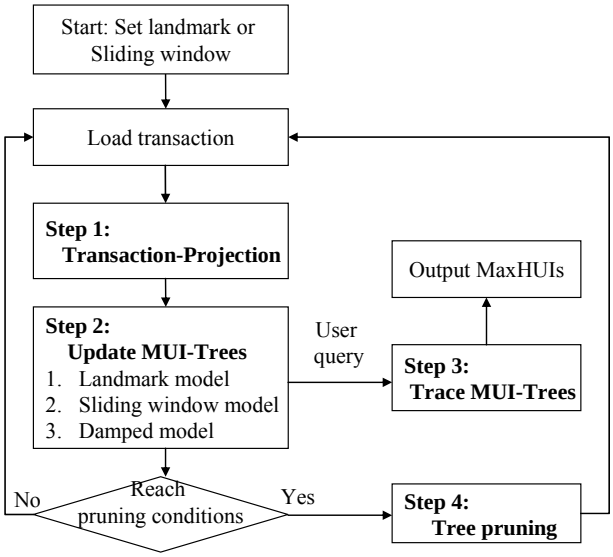


Figure 1. The flowchart of the proposed framework GUIDE.

Definition 4. (Transaction-projection.) Without loss of generality, we assume that all items in transactions are sorted in a fixed order, e.g. the alphabetical order. Assume a transaction $\{(i_1, q_1)\ (i_2, q_2)\ ...\ (i_n, q_n)\}$ arrives into a data stream. First, the transaction is projected into all its postfixes, i.e., $\{(i_1, q_1)\ (i_2, q_2)\ ...\ (i_n, q_n)\}$, $\{(i_2, q_2)\ (i_3, q_3)\ ...\ (i_n, q_n)\}$, ..., and $\{(i_n, q_n)\}$. Then each postfix is projected again to get all its prefixes, such as $\{(i_2, q_2)\ (i_3, q_3)\ ...\ (i_n, q_n)\}$ is projected to $\{(i_2, q_2)\ (i_3, q_3)\ ...\ (i_n, q_n)\}$, $\{(i_2, q_2)\ (i_3, q_3)\ ...\ (i_{n-1}, q_{n-1})\}$, ..., and $\{(i_2, q_2)\}$. At the same time, the utility of each projection is calculated. These sub-itemsets are called *projections*.

The generated projections are collected in a stack in order. By the pushed order of the projections, they will be popped by the order of $\{(i_1, q_1)\}$, $\{(i_1, q_1)\ (i_2, q_2)\}$, ...,$\{(i_1, q_1)\ (i_2, q_2)\ ...\ (i_n, q_n)\}$, $\{(i_2, q_2)\}$, $\{(i_2, q_2)\ (i_3, q_3)\}$, ..., $\{(i_2, q_2)\ (i_3, q_3)\ ...\ (i_n, q_n)\}$, ..., $\{(i_n, q_n)\}$. This order will help the update of MUI-Tree.

Example 1. Consider the data stream *DS* and utility table in Tables 1 and 2. Assume the landmark is set at time 0. At time 2, the first transaction $Tid_1 = \{(C, 12)\ (E, 2)\}$ arrives to *DS*. The utility of the transaction is first added to *TotalU* of *DS*. $GUIDE_{LM}$ first performs transaction-projection to get the projections of Tid_1. In this example, the projections of Tid_1 are {C}, {E} and {CE}, and their utilities are 12, 10 and 22, respectively. □

Inevitably, transaction-projection may result in some pattern loss. GUIDE provides the balanced results between runtime and accuracy, which satisfies the needs of users. Further evaluations and discussions on this issue are given later in our experiments.

Procedure *Transaction-projection*
Input: A transaction Tid_k
Output: A set of projections of Tid_k, denoted as $Proj_k$

1. Stack $Proj_k = \phi$
2. **while** $Tid_k \neq \phi$
3. add $\{Tid_k, u_{Tid_k}\}$ into $Proj_k$
4. $Tid_{k\_temp} = Tid_k$ //*a copy of* Tid_k
5. **while** $Tid_{k\_temp} \neq \phi$
6. prune the last item of Tid_{k_temp}
7. add Tid_{k_temp} into $Proj_k$
8. **end while**
9. prune the first item of Tid_k
10. **end while**

Figure 2. The procedure of transaction-projection.

Subsequently, the projections are maintained in a tree structure, called *MUI-Tree* (*Maximal high Utility Itemset Tree*). Here, we proposed three methods for different models of data stream mining, i.e., $GUIDE_{LM}$ for landmark model, $GUIDE_{SW}$ for sliding window model, and $GUIDE_{TF}$ for time fading model. The trees for different models are described from the subsections 4.1 to 4.3, respectively. When users give queries to the system, MaxHUIs are checked by the proposed *bottom-up tracing strategy*. Then the MaxHUIs are output. Finally, when the pruning conditions are reached, a pruning strategy will be performed for decreasing the memory usage of MUI-Tree. The tracing and pruning of MUI-Trees are introduced in the subsections 4.4 to 4.5.

4.1 $GUIDE_{LM}$: The Proposed Method for Landmark Model

In this subsection, we describe the processes of GUIDE for the landmark model. The method for landmark model mining is called $GUIDE_{LM}$. The procedure of $GUIDE_{LM}$ is shown in Figure 3. After projecting the transaction, the projections are inserted into the MUI_{LM}*-Tree*. First, we define the elements in MUI_{LM}-Tree as follows.

Definition 5. (The elements in MUI_{LM}-Tree.) A MUI_{LM}-Tree is composed of *nodes* and *links*. A projection is stored in a corresponding node in a MUI_{LM}-Tree. The node of MUI_{LM}-Tree, denoted as <*id*: *utility*: *time*>, is a triple that consists of the identity of the itemset *X*, the utility of *X*, and the time *X* was added into MUI_{LM}-Tree. Moreover, each node has two kinds of links: the first one is *parent-link*, which points to its parent node, and the second one is *children-link*, which points to its children nodes.

The procedure of MUI_{LM}-Tree updating is shown in Figure 4. Each projection is checked to see whether its corresponding node exists in the MUI_{LM}-Tree. If the node does not exist, it is created; otherwise, the node is updated by accumulating the utility of the

projection.

Algorithm ***GUIDE$_{LM}$***
Input: A data stream *DS*, a pre-defined utility table and a user-defined minimum utility threshold *MinU*
Output: A list of MaxHUIs
1. Initialization: *MUI$_{LM}$-Tree* = ϕ and *TotalU* = 0
2. **while** a new transaction *Tid$_k$* arrives into *DS*
3. *TotalU* = *TotalU* + *u*(*Tid$_k$*)
4. *Proj$_k$* = ***Transaction-projection***(*Tid$_k$*)
5. **for** each projection $p \in Proj_k$
6. ***MUI$_{LM}$-Tree_updating***(*p*, *MUI$_{LM}$-Tree*)
7. **end for**
8. **end while**
9. **if**(*user_request* = true)
10. set a pointer *pt* which points to the leftist leaf node of *MUI$_{LM}$-Tree*
11. *temp_list* = ***bottom-up_tracing***(*MUI$_{LM}$-Tree*, *MinU*, *pt*)
12. output MaxHUIs in *temp_list*
13. **end if**

Figure 3. The procedure of GUIDE$_{LM}$.

Procedure ***MUI$_{LM}$-Tree_updating***
Input: a projection *p* and a tree structure *MUI$_{LM}$-Tree*
1. trace *MUI$_{LM}$-Tree* to find the node of *p*
2. **if** the node does not exist
3. create a new node <*i*: *p.util*: t_{now}> in *MUI$_{LM}$-Tree* //*p.util is the utility of p*
4. **else**
5. modify the node's utility by adding *p.util*
6. **end if**

Figure 4. The procedure of MUI$_{LM}$-Tree updating.

Example 2. Let us continue the example 1. First, the projection {C} is checked in the MUI$_{LM}$-Tree. Since MUI$_{LM}$-Tree is null now, all corresponding nodes of these projections are added. The node <{C}: 12: 2> is first added as a child node of the root in the MUI$_{LM}$-Tree. Then the projection {CE} is checked, and the node <{E}: 22: 2> is added as a child node of <{C}: 12: 2>. Finally the projection {E} is checked, and the node <{E}: 10: 2> is added as another child node of the root. The MUI$_{LM}$-Tree now is shown in Figure 5 (a). At this time, *TotalU* of *DS* is 22. When the second transaction {(B, 6) (D, 1) (E, 1)} arrives at time 4, it is projected into six projections {B}, {BD}, {BDE}, {D}, {DE} and {E} with the utilities 60, 66, 71, 6, 11 and 5, respectively. As the same processes mentioned previously, two branches <{B}: 60: 3>→<{D}: 66: 3>→<{E}: 71: 3> and <{D}: 6: 3>→<{E}: 11: 3> are created and inserted into MUI$_{LM}$-Tree sequentially. When

we check the corresponding node of the projection {E}, we find that a node <{E}: 10: 2> is already in the MUI_{LM}-Tree. So we just update the node by adding the utility of the projection to this node. After updating, the utility of this node becomes 15. The resulted MUI_{LM}-Tree is shown in Figure 5 (b). The current *TotalU* of *DS* is 22+71=93. Figure 5 shows the construction of the MUI_{LM}-Tree with adding all transactions of the example database *DS* in turn. The gray nodes in the MUI_{LM}-Trees stand for the nodes were updated or created at that time. □

We can see that in this example, the popped order of projections makes the branches in the MUI_{LM}-Tree be updated by the order of depth-first search. Thus the update of the MUI_{LM}-Tree does not need too many additional overhead to find the corresponding nodes of projections.

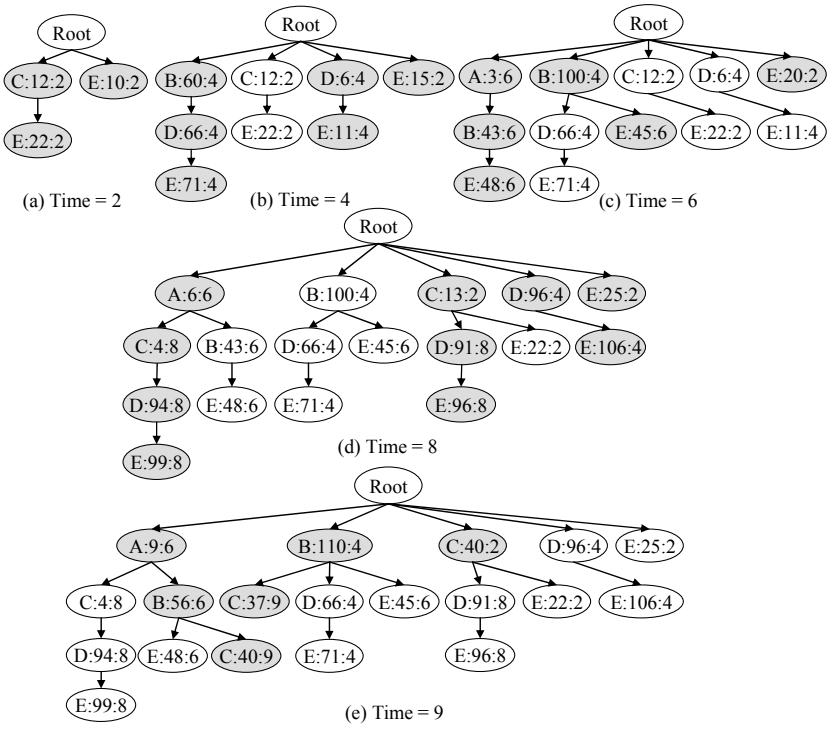


Figure 5. The updating of MUI_{LM}-Tree for landmark model.

4.2 GUIDE$_{SW}$: The Proposed Method for Sliding Window Model

Mining patterns from data streams by the sliding window model means to find the patterns from the valid transactions in the window. When the window slides, new valid transactions are added into the window and old invalid transactions should be pruned. By the demand of different applications, the window can be classified to two types as follows. 1) *Time-sensitive window*: The window for a fixed period of time, such as one month; 2) *Transaction-sensitive window*: The window for fixed size of transactions, such as ten thousand transactions. In this paper, we discuss about the time-sensitive window. Note that the proposed framework can fit both types of windows. For dealing with the case of transaction-sensitive window, the proposed method only needs to replace the time by the TID for the transactions.

We show the updating of MUI-Tree for the sliding window model. The MUI-Tree is called MUI$_{SW}$-Tree and the method for sliding window mining is called GUIDE$_{SW}$. The procedure of GUIDE$_{SW}$ is shown in Figure 6. The main differences of GUIDE$_{SW}$ and GUIDE$_{LM}$ are that the MUI$_{SW}$-Tree further maintains a *time-slot queue* in each node for capturing individual utilities of itemsets in different time or transactions. Moreover, MUI$_{SW}$-Tree keeps *time-links* for linking the nodes involving in the same time.

Definition 6. (Time-slot queue in MUI$_{SW}$-Tree.) A *time-slot queue* is a queue for time-slots. A *time-slot* in a time-slot queue, denoted as <*utility*, *time*>, is a pair that consists of the utility of the itemset and its corresponding time. When a new time-slot is added into the time-slot queue, it is inserted to the end of the queue; on the other hand, when an old time-slot is out-of-date, it will be removed from the beginning of the queue.

Algorithm *GUIDE$_{SW}$*
Input: A data stream *DS*, a pre-defined utility table, a user-defined minimum utility threshold *MinU* and a user-specified window size t_{SW}
Output: A list of MaxHUIs
1. Initialization: MUI_{SW}-*Tree* = ϕ and *TotalU* = 0
2. **while** a new transaction Tid_k arrives into *DS*
3. *TotalU* = *TotalU* + *u*(Tid_k)
4. $Proj_k$ = ***Transaction-projection***(Tid_k)
5. **for** each projection $p \in Proj_k$
6. ***MUI$_{SW}$-Tree_updating***(p, MUI_{SW}-*Tree*)
7. **end for**
8. **end while**
9. set *time-links* to link all bottom modified nodes in different branches
10. **if**(*user_request* = true)
11. set a pointer *pt* which points to the leftist leaf node of MUI_{SW}-*Tree*
12. *temp_list* = ***bottom-up_tracing***(MUI_{SW}-*Tree*, *MinU*, *pt*)
13. output MaxHUIs in *temp_list*
14. **end if**

Figure 6. The procedure of GUIDE$_{SW}$.

Procedure *MUI_{SW}-Tree_updating*
Input: a projection p, a tree structure *MUI_{SW}-Tree*, a user-specified window size t_{sw} and a current time point t_{now}
1. trace *MUI_{SW}-Tree* to find the node of p
2. **if** the node does not exist
3. create a new node <i: *p.util*: t_{now}> in *MUI_{SW}-Tree* //*p.util is the utility of p*
4. **else**
5. modify the node's utility by adding *p.util*
6. **end if**
7. add a new time-slot <*p.util*, t_{now}> to the *time-slot queue* of this node
8. **while** there exists a time point $t_{old} < t_{now} - t_{sw}$
9. trace the *time-links* related to t_{old} and remove the time-slots corresponding to t_{old}
10. **end while**

Figure 7. The procedure of MUI_{SW}-Tree updating.

Definition 7. (Time-link.) A set of *time-links* corresponding to the time point t_p connect the nodes those satisfy the following conditions: 1) Each node has the time-slot with the time point t_p; 2) each node linked by the links is the most bottom node having the time-slot with the time point t_p in the branch.

Here, we use an example to explain the process of updating the MUI_{SW}-Tree, which is shown in Figure 7.

Example 3. Let us continue the example 1. Assume the window size is set to 5 time units. At time 2, the first transaction Tid_1 = {(C, 12) (E, 2)} arrives to *DS*. The utility of the transaction is first added to the *TotalU* of *DS*. Tid_1 is projected to {C}, {E} and {CE} with the utilities 12, 10 and 22, respectively. The projections are added into the MUI_{SW}-Tree by similar method to $GUIDE_{LM}$. Three nodes are added into the tree: <C: 12> with a time-slot <12, 2>, <E: 22> with a time-slot <22, 2> and <E: 10> with a time-slot <10, 2>. Since <E: 22> and <E: 10> are the most bottom nodes among the two modified branches with time 2, they are linked by the time-links of time 2. The results are shown in Figure 8 (a). At time 4, the second transaction arrives to *DS*. Three branches are modified: <B: 60>→<D: 66>→<E: 72>, <D: 6>→<E: 11> and <E: 15>. For the node <E: 15>, since there is a time-slot <10, 2> in its queue, a new time-slot <5, 4> is added to the end of the queue. Finally, time-links are added to link the corresponding nodes of time 4. The results are shown in Figure 8 (b). The third transaction is inserted into the MUI_{SW}-Tree at time 6 as shown in Figure 8 (c). (For simplicity, the time-links are not shown in the figures after time 4.)

At time 7, the information about time 2 should be slid out from the window. The time-links about time 2 are traced and two branches with the leaf nodes <E: 22> and <E: 10> are found. In the found nodes, the time-slots about time 2 are removed from the nodes and the utilities of the time-slots are subtracted from the node utilities. After removing the time-slots, the utilities of the nodes <C: 12>, <E: 22> and <E: 20> become <C: 0>, <E: 0> and <E: 10>. Since there is no time-slot in the two nodes <C: 0> and <E: 0>, they are removed from the MUI_{SW}-Tree. The results are shown in Figure 8 (d). The transactions

are added and removed by the processes as shown in Figure 8. □

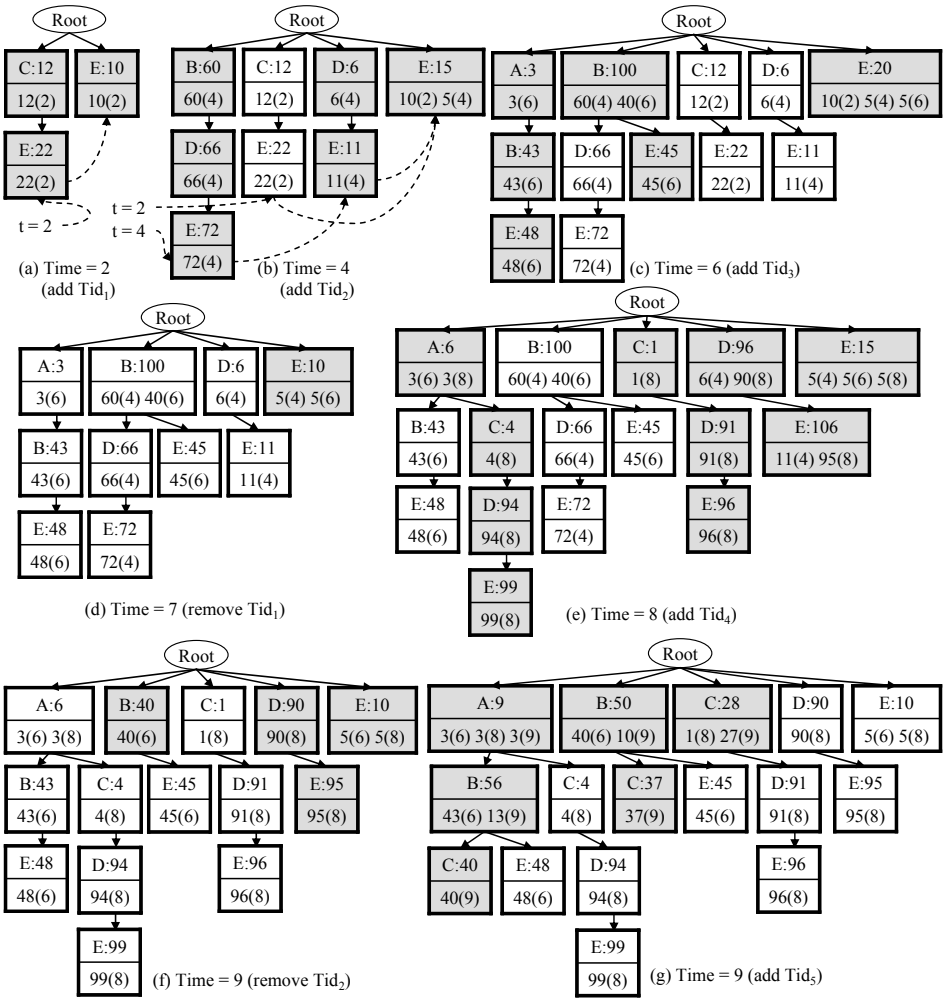


Figure 8. The updating of MUI_{SW}-Tree for sliding window model.

4.3 GUIDE$_{TF}$: The Proposed Algorithm for Time Fading Model

In the time-fading model, the data is also accumulated from the landmark time to the present and the results are generated from the whole data. However, user-defined *time decay functions* are given to decrease the weight of old itemsets. The proposed framework GUIDE can also apply to the time fading model. First, we define the *time-fading utility* in our time fading model by the most common time decay function, i.e., exponential decay.

Definition 8. (Time-fading utility with exponential decay.) Assume the *decay factor* is α and the transactions related to the itemset X from t_{LM} are Tid_1, Tid_2, …, Tid_n appeared at time t_1, t_2, …, t_n. The *time-fading utility* of X at t_n is denoted and defined as the following equation:

$$u(X, t_n) = \sum_{k=1}^{n} \left(u(X, Tid_k) \times \alpha^{(t_n - t_k)} \right)$$

Both the structures of MUI$_{SW}$-Tree and MUI$_{LM}$-Tree can be applied to acquire the time-fading utilities of itemsets. The first one for MUI$_{SW}$-Tree is to maintain the unit utilities at different time in time-slots, like the MUI$_{SW}$-Trees shown in Figure 8. When users give queries for MaxHUIs, the utility of each node is calculated by Definition 8 when it is traced. For example, assume the decay factor α is 0.9. At time 9, the time-fading utility of the node <B: 50> in Figure 8 (g) (the first node in the second branch) is calculated as $u(\{B\}, t_9) = 40 \times 0.9^{(9-6)} + 10 \times 0.9^{(9-9)} = 29.16 + 10 = 39.16$. By this method, GUIDE$_{TF}$ can find MaxHUIs in the time fading model.

On the other hand, the second method for MUI$_{LM}$-Tree is only maintaining one utility value in each node. However, at each time point, all nodes are updated by multiplying the decay factor. The main process is shown in Figure 9.

The two methods provide a trade-off between runtime and memory usage. For the first method that maintains the information in the MUI$_{SW}$-Tree, it needs more computations when users give queries since it only maintains time and utilities in individual time points. Current utilities will be calculated real-time before output. Moreover, it needs more memory space to save the time-slot queues in the nodes. On the other hand, the second method that maintains the information in the MUI$_{LM}$-Tree does not need to compute the node utilities after tracing the nodes. Thus it needs less response time after users query the system. Moreover, it needs less memory space since it does not need to maintain the time-slot queues in the nodes. However, this method needs to update all nodes after each time point, which involves huge computational cost.

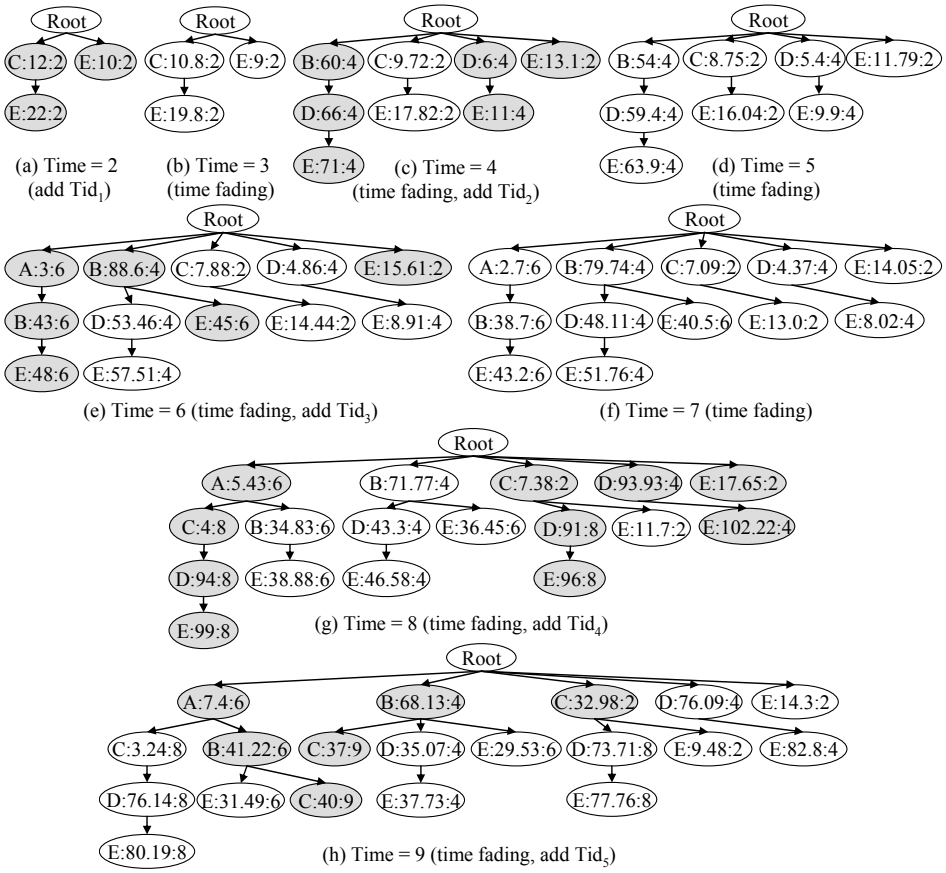


Figure 9. The updating of MUI_{LM}-Tree for time fading model.

4.4 Generating MaxHUIs from MUI-Trees

Whenever if a user queries for the MaxHUIs with a specified minimum utility threshold, the MUI-Trees will be traced and the MaxHUIs are output. The generation of MaxHUIs consists of two main processes: 1) Tracing MUI-Trees and get HUIs in the nodes, and 2) check MaxHUIs within the set of HUIs. Since there is no *downward closure* [1, 23, 28] in the utility mining, that is, the subsets of a HUI may not be a HUI, all nodes in MUI-Tree should be traced to grab all HUIs. Thus the traditional method for the first process is to trace the MUI-Tree by the *top-down tracing strategy* (abbreviated as *TDT strategy*; also

known as *depth first tracing strategy*). After getting the HUIs, the second process is applied to verify the MaxHUIs within the HUIs.

The main drawback of the above method is that it needs to check all nodes in MUI-Trees. When there are too many tree nodes, this process will become time-consuming to the users. In view of this, we proposed a *bottom-up tracing strategy*, abbreviated as *BUT strategy*, which is described as below.

Strategy 1. Bottom-up tracing. Assume the minimum utility threshold is *MinU*. Each branch in the MUI_{LM}-Tree is traced from the leaf node. First, a pointer is set to the leftist leaf node. The nodes in the branch are checked from the leaf to root. While checking a node *N*, if *N*'s utility is larger than or equal to *MinU*, the itemset that the node represents is put to a list for HUIs. Besides, the ancient nodes from *N* to the root are also labeled as *checked*, i.e., these nodes do not need to be checked since they are impossible to be *maximal*. If *N*'s utility is less than *MinU*, the pointer goes to the parent node of *N*. After all branches are checked, the process is finished. □

Example 4. Let us continue the example 2. Take the MUI_{LM}-Tree in Figure 5 (e) as an example. Assume there is a user gives a query *MinU*=70 after time 9. The MUI_{LM}-Tree is traced as follows. First, a pointer *pt* is set to point to the leftist leaf node <E: 99: 8>. Since the utility of the node is larger than $TotalU \times MinU$, i.e., 99 > 70, the itemset {ACDE} is put to the HUI list. Then the parent nodes of this node, i.e., <D: 94: 8>, <C: 4: 8> and <A: 9: 6>, are set as *checked*. Then *pt* goes to the leaf node of the next branch, i.e., <E: 48: 6>. Since the utility of this node is less than 70, *pt* goes to the parent node of this node, i.e., <B: 56: 6>. Because the utility of this node is also less than 70, *pt* goes to its parent node <A: 9: 6>. However, since this node is marked as *checked*, it does not be checked. *Pt* goes to the leaf node of the next branch, i.e., <C: 40: 9>. The MUI_{LM}-Tree is recursively checked in the same way. Finally, four HUIs are added to the temp list: {ACDE}, {BDE}, {CDE} and {DE}. After verifying the HUIs, two MaxHUIs {ACDE} and {BDE} are generated. □

If we generate HUIs by the top-down tracing strategy, all the HUIs in the same branch will be generated. Obviously, only the HUIs in the deeper nodes of the MUI-Trees could be maximal. The bottom-up tracing strategy avoids generating the HUIs from the same branch. It reduces the number of HUIs that should be checked, which saves both runtime and memory usage.

4.5 The Pruning Strategy for MUI-Trees

Infinite data from data streams may not be maintained in MUI-Trees due to the resource constraint. Therefore, we propose a pruning method for MUI-Trees to prevent the memory exhaustion. The pruning processes are performed in the following two conditions: 1) Time-oriented: The elapsed time is more than a user-defined time threshold, such as one hour; 2) Memory-oriented: The memory usage of the MUI-Tree is more than a user-defined memory usage threshold, such as 70% of available memory. The main purpose of our pruning strategy is *to keep the nodes with later time and larger utilities,*

but to prune the nodes with earlier time and smaller utilities. We first define an essential factor called *the ratio of pass time* for the pruning strategy.

Definition 9. (The ratio of pass time.) Assume that t_{now} is the current time, t_{old} is the oldest valid time, t_{SW} is the window size and t_x is the time of the node x. The *ratio of pass time* of x is denoted and defined as $\mathrm{PT}(x) = \frac{t_{now} - t_x}{t_{now} - t_{old}}$.

In landmark and time fading models, t_{old} is the landmark time; in sliding window model, t_{old} is t_{now}-t_{SW}. In $\mathrm{MUI_{LM}}$-Tree, t_x is the time of the node; in $\mathrm{MUI_{SW}}$-Tree, t_x is the smallest time point recorded in the time-slot queue, i.e., the time of the first time-slot in the queue. Without loss of generality, we define the ratio of pass time of t_{old} and t_{now}, i.e., $\mathrm{PT}(t_{old})$ and $\mathrm{PT}(t_{now})$, as 1 and 0, respectively.

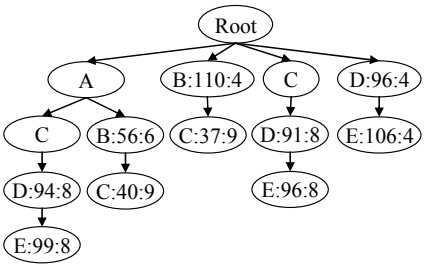
An example is shown as follows. Assume that the landmark model and $\mathrm{MUI_{LM}}$-Tree are applied. The landmark time is 0, the current time is 10 and the itemsets {A} and {B} appeared in the data stream at time 2 and 7, respectively. The ratio of pass time of {A} and {B} can be calculated as PT({A}) = (10-2)/(10-0) = 0.8 and PT({B}) = (10-7)/(10-0) = 0.3.

Strategy 2. The pruning strategy for MUI-Trees. Assume that a user-specified *pruning factor* is β. The *pruning threshold* of a node x is defined as $PT(x) \times \beta \times TotalU$. When the pruning process is requested, each branch in MUI-Tree is checked from the leaf to root. The node x will be pruned if its utility is less than the pruning threshold. If x is a leaf node, it is deleted; otherwise, x and its ID is kept; only its utility and time are removed. For $\mathrm{MUI_{SW}}$-Tree, x's time-slot queue is also removed. □

The pruning threshold is determined by the time when the node was added into the MUI-Tree. Thus by the proposed pruning strategy, the pruning threshold of a node is smaller if the node was added into MUI-Tree later; otherwise the threshold is larger.

Example 5. Let us continue the example 4. Assume the current time is at time 10, the landmark time is at time 0, the pruning factor β is 0.5, and the current tree is the $\mathrm{MUI_{LM}}$-Tree shown in Figure 5 (e) with the *TotalU* = 280. Each branch in the $\mathrm{MUI_{LM}}$-Tree is checked from the leaf to root. For the first node <{E}: 99: 8>, the pruning threshold of this node is $0.5 \times 280 \times ((10\text{-}8)/(10\text{-}0)) = 28$. Since the utility of this node is larger than the threshold, the node is kept. Then the second node <{D}: 94: 8> is kept since the utility of this node is also larger than the pruning threshold, i.e., 94 > 28. On the other hand, the third node <{C}: 4: 8> should be delete since its utility is less than the pruning threshold, i.e., 4 < 28. However, this node is an internal node. If it is pruned directly, the children nodes will have no parent node. Thus, only its utility and time are removed. When checking the node in the last branch <{E}: 25: 2>, the pruning threshold of this node is $0.5 \times 280 \times ((10\text{-}2)/(10\text{-}0)) = 110$. This node is pruned since its utility is less than the pruning threshold and it is a leaf node. After the pruning processes, the resulted $\mathrm{MUI_{LM}}$-Tree is shown in Figure 10 (a). Comparing with the original $\mathrm{MUI_{LM}}$-Tree in Figure 5 (e), 47% nodes are modified and 32% nodes are deleted. Since the pruning threshold mainly depends on the time of the node, to reduce the computational cost, we also provide a *pruning threshold table* which records the pruning threshold for each time

point as shown in Figure 10 (b). □



(a) The pruned MUI_{LM}-Tree

Time	Pruning threshold
1	126
2	110
3	96
4	84
5	70
6	56
7	42
8	28
9	14
10	0

(b) Pruning threshold table

Figure 10. The example of the pruning strategy for MUI-Trees.

Table 3. Parameter settings of synthetic datasets.

Parameter	Description	Default
D	Number of transactions in data stream	10k
T	Average size of the transactions	5
N	Number of distinct items	1,000
q	Number of purchased items in transactions	1~5
pr	Unit price of each item	1~1,000

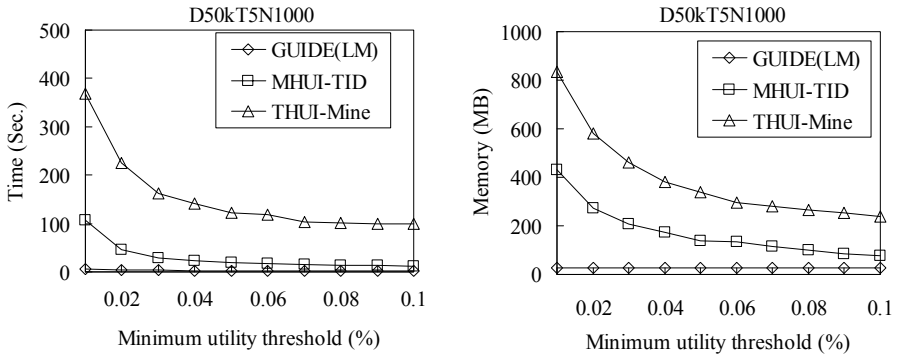
5 Experimental Evaluation

In this section, we evaluate the performance of proposed algorithms. The experiments were preformed on a PC with 3.4 GHz CPU, 4 GB memory and the operating system is Microsoft Windows 7 64-bit. All algorithms are implemented in Java. The experiments are conducted by the synthetic datasets generated from the data generator in [28]. Parameters and default settings for the synthetic datasets are shown in Table 3. The performance of the proposed methods was compared with two state-of-the-art algorithms for mining high utility itemsets from data streams, i.e., MHUI-TID [20] and THUI-Mine [27].

5.1 Performance on the Algorithms for Landmark Model

First, we show the performance of $GUIDE_{LM}$ for landmark model. Figure 11 shows the runtime and memory usage of the compared algorithms under varied minimum utility

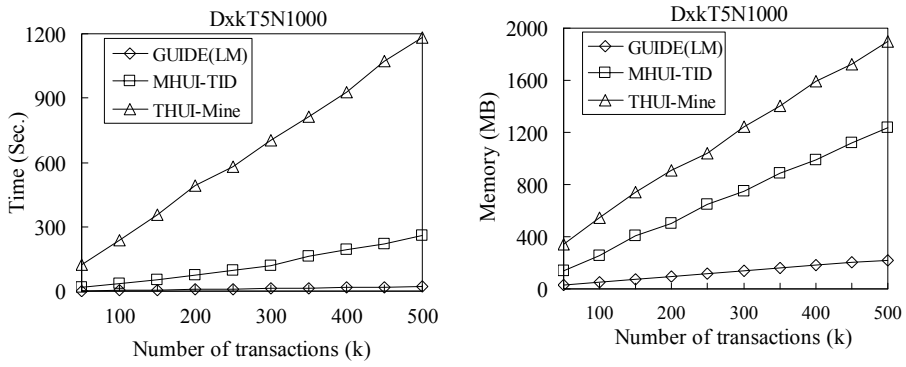
thresholds. The tested dataset for the experiment is D50kT5N1000. For MHUI-TID and THUI-Mine, since they are designed for the sliding window model, we set the window size to the data size to capture all data from the landmark time point. In Figure 11 (a), the runtime of $GUIDE_{LM}$ is the best, followed by MHUI-TID, and THUI-Mine is the worst. This is because that comparing with the two level-wise-based methods, $GUIDE_{LM}$ directly maintains potential MaxHUIs in the MUI_{LM}-Tree and generates patterns by the efficient bottom-up tracing strategy. When the minimum utility threshold is low, only $GUIDE_{LM}$ can generate the results in few seconds, which fits the speed requirements of data stream mining. In Figure 11 (b), the memory usage of $GUIDE_{LM}$ remains the same since it maintains MUI_{LM}-Trees without considering the threshold. The two algorithms MHUI-TID and THUI-Mine need to maintain the proposed structures, candidate itemsets and the transactions in the windows (in landmark model, all transactions are maintained), thus their memory usage is much worse than $GUIDE_{LM}$.



(a) Runtime (b) Memory usage

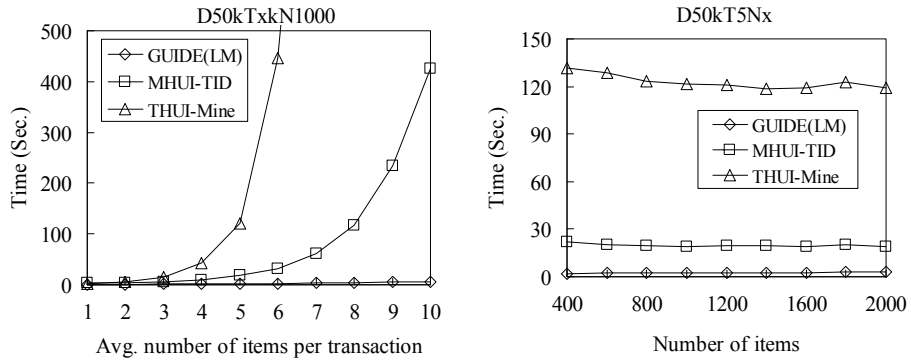
Figure 11. Evaluation on varying minimum utility threshold

Next, we show the scalability of the compared algorithms. The results are shown in Figure 12. The minimum utility threshold is set to 0.05%. It can be observed from Figure 12 (a) that all compared algorithms have good scalability. All of them increase linearly when the number of transactions in the data streams is increased. It can also be seen that $GUIDE_{LM}$ is the most efficient algorithm among the compared algorithms. Figure 12 (b) shows the memory usage of the three algorithms. $GUIDE_{LM}$ performs better than MHUI-TID and THUI-Mine since it does not need to store all transactions in the window.



(a) Runtime (b) Memory usage

Figure 12. Scalability of $GUIDE_{LM}$



(a) Varying number of transactions (b) Varying number of items

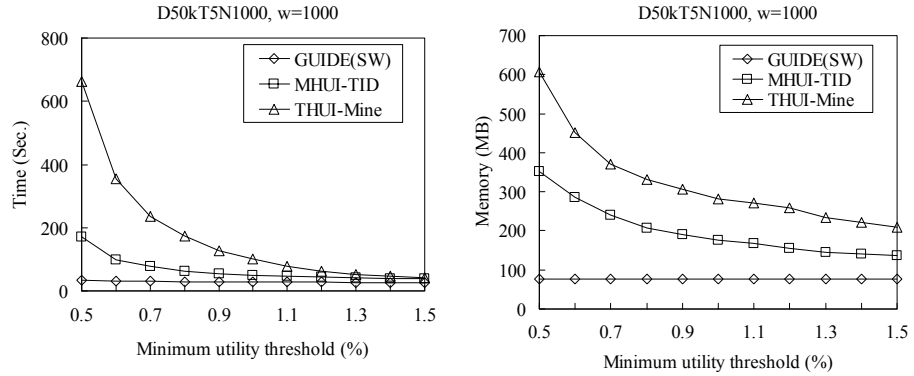
Figure 13. Performance on varied parameters

Next, the performance of the algorithms on different parameters is evaluated. Figure 13 (a) shows the results of varying number of items per transaction (T). We can see the runtime of the algorithms increases exponentially with the increasing of T. $GUIDE_{LM}$ is the most stable algorithm among the three algorithms. The reason is that unlike the level-wise-based results those need time-consuming processes for generating the patterns, $GUIDE_{LM}$ only needs to trace the MUI_{LM}-Tree. Since the proposed bottom-up tracing strategy effectively reduces the number of traced nodes when T is large, the runtime of $GUIDE_{LM}$ outperforms MHUI-TID and THUI-Mine. (The discussions about the BUT strategy are given in the subsection 5.3.)

Figure 13 (b) shows the runtime of varying number of items (N) for the compared algorithms. Generally speaking, the runtime of the algorithms is not influenced by this parameter. (If N is large enough, the dataset will be too sparse and no pattern will be generated.) In this experiment, $GUIDE_{LM}$ is still the best performer among the algorithms.

5.2 Performance of the Algorithms for Sliding Window Model

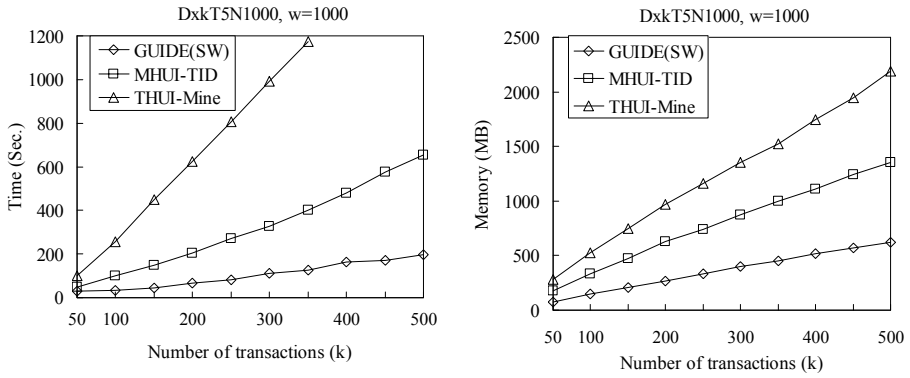
In this subsection, the performance evaluation about the algorithms for the sliding window model is presented. First, we show the performance comparison under varied minimum utility thresholds. The tested dataset is D50kT5N1000. The results are shown in Figure 14. It can be observed that the performance of $GUIDE_{SW}$ is the best, followed by MHUI-TID, and THUI-Mine be the worst. Besides, in Figure 14 (a) and (b), $GUIDE_{SW}$ performs worse than $GUIDE_{LM}$ on not only runtime but also memory usage. It is because that the update of all time-slots should be dealt with in $GUIDE_{SW}$.



(a) Runtime (b) Memory usage

Figure 14. Evaluation on varying minimum utility threshold

Next, we show the scalability of the compared algorithms. The results are shown in Figure 15. The minimum utility threshold is set to 1%. In Figure 15 (a) and (b), we can see that the proposed algorithm $GUIDE_{SW}$ has good scalability. It surpasses the compared algorithms MHUI-TID and THUI-Mine by about 3 times and 10 times, respectively. Comparing to $GUIDE_{LM}$, the memory usage of $GUIDE_{SW}$ is worse since the time-slot queues need to be maintained in the MUI_{SW}-Trees.

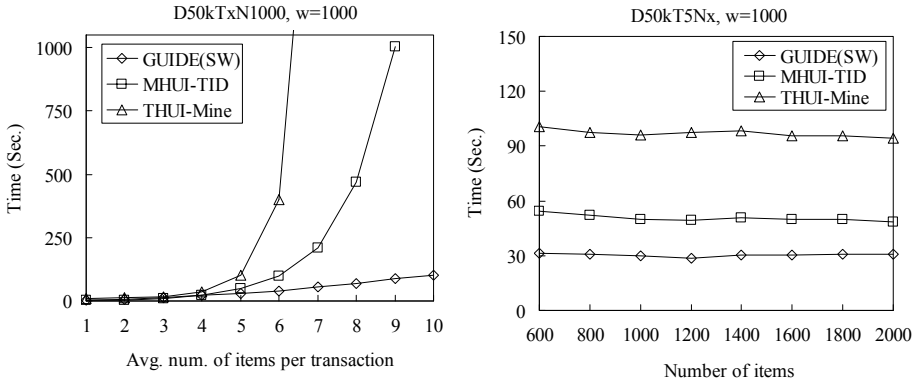


(a) Runtime (b) Memory usage

Figure 15. Scalability of $GUIDE_{SW}$

Subsequently, we show the performance of the compared algorithms with different parameters. Figure 16 (a) is the results of varying number of items per transaction (T). It can be observed that similar to Figure 13 (a), the runtime of the algorithms increases exponentially with the increasing of T. Among the three algorithms, $GUIDE_{SW}$ is the most stable algorithm. The reason is that $GUIDE_{SW}$ does not need time-consuming processes for generating the patterns. Figure 16 (b) shows the runtime of varying number of items (N) for the compared algorithms. Generally, similar to Figure 13 (b), the runtime of the algorithms is not influenced by this parameter. In this experiment, $GUIDE_{SW}$ still outperforms the two compared algorithms.

In the experiments, we can see that not only the runtime of $GUIDE_{LM}$ outperforms that of $GUIDE_{SW}$ but also those of MHUI-TID and THUI-Mine in landmark model outperform those in sliding window model. The reason is that although the methods for landmark model need to process the whole data from the landmark time, those for sliding window model need to perform the time-consuming processes for updating information when windows slide at each time point.



(a) Varying avg. number of items per transactions (b) Varying number of items

Figure 16. Performance on varied parameters

5.3 Effects of Bottom-Up Tracing Strategy

In this subsection, we show the effects of the bottom-up tracing strategy. We analyze the number of checked nodes for different tracing strategies by the measurement *node reduction ratio*, abbreviated as *NRR*. The equation of NRR is shown as follows.

$$\text{Node reduction ratio} = \frac{\#\text{Nodes checked by the bottom up tracing strategy}}{\#\text{Nodes checked by the top down tracing strategy}} \times 100\%$$

The results on varying minimum utility thresholds and varying average number of items per transactions (T) are shown in Table 4 (a) and (b), respectively. The tested algorithm is GUIDE_{LM}. The tested dataset for the first experiment is D50kT5N1000. In Table 4 (a), we can observe that the number of traced nodes by the traditional TDT strategy remain the same. The reason is that all nodes in the MUI-Tree are traced by the TDT strategy. On the other hand, the number of traced nodes by the BUT strategy increases with the increasing minimum utility thresholds. The reason is that the lower the minimum utility threshold, the easier a node in a branch satisfies it. In other words, when the minimum utility threshold is lower, the possibility of the nodes near the leaf nodes pass the threshold is larger. In average, NRR is about 68.33%, that is, about 1/3 nodes can be skipped during the tracing process.

The tested datasets for the second experiment are D50kT*x*N1000 and the minimum utility threshold is set to 0.05%. In Table 4 (b), it can be observed that NRR decreases with the increase of T. When T=1, NRR is 96.88%; however when T=10, NRR is 38.54%.

This is because that when T is increased, the lengths of branches in MUI-Trees are longer. By the BUT strategy, the tracing of branches may terminate at some internal nodes. If the minimum utility threshold is small enough, the tracing of MUI-Tree can be finished after tracing all leaf nodes.

Table 4. Number of nodes by different tracing strategies.

(a) Varying minimum utility thresholds

MinUtil	#Nodes by BUT	#Nodes by TDT	NRR
0.01 %	190,553	323,467	58.91 %
0.02 %	206,036	323,467	63.69 %
0.03 %	215,181	323,467	66.52 %
0.04 %	221,042	323,467	68.34 %
0.05 %	224,933	323,467	69.54 %
0.06 %	227,698	323,467	70.39 %
0.07 %	229,640	323,467	70.99 %
0.08 %	230,911	323,467	71.39 %
0.09 %	231,845	323,467	71.68 %
0.10 %	232,414	323,467	71.85 %

(b) Varying avg. number of items per transactions (T)

T	#Nodes by BUT	#Nodes by TDT	NRR
1	11,252	11,614	96.88 %
2	37,098	40,057	92.61 %
3	84,659	99,059	85.46 %
4	149,404	194,710	76.73 %
5	224,933	323,467	69.54 %
6	324,361	523,394	61.97 %
7	426,186	756,278	56.35 %
8	530,138	1,042,222	50.87 %
9	479,857	1,131,084	42.42 %
10	675,091	1,751,787	38.54 %

5.4 Quality of Patterns

In this subsection, we show the quality of found patterns. We analyze the patterns in two aspects, namely precision and *pattern reduction ratio* (abbreviated as *PRR*). The equations of the two measurements are shown as follows.

$$\text{Precision} = \frac{|\,\text{MaxHUI} \cap \text{Actual maximal HUI}\,|}{|\,\text{MaxHUI}\,|} \times 100\%$$

$$\text{Pattern reduction ratio} = \frac{|\,\text{MaxHUI}\,|}{|\,\text{HUI}\,|} \times 100\%$$

The results about varying minimum utility thresholds and database sizes are shown in Table 5 (a) and (b), respectively. For Table 5 (a), the tested dataset is D50kT5N1000; for Table 5 (b), the minimum support is set to 0.05%. The testing algorithm is $GUIDE_{LM}$. In Table 5, overall pattern reduction ratio is about 19%. When the minimum utility threshold is 0.01% and the size of database is 50k, $GUIDE_{LM}$ presents only 1047 MaxHUIs among 16620 HUIs. It effectively reduces 93.7% redundant patterns. Moreover, we can observe that the number of MaxHUIs and HUIs decrease when the minimum utility threshold increases. With the decreasing of the number of HUIs, the PRR raises since the number patterns derived from the same branches of MUI-Trees are decreased. The precision of GUIDE is not 100% since the transaction-projection provides incomplete information for GUIDE. The average precision is about 83%.

Table 5. Quality of Patterns.

(a) Varying minimum utility thresholds

MinUtil	#MaxHUI	#HUI	PRR	Prec.
0.01 %	1,047	16,620	6.30 %	87.68 %
0.02 %	899	9,340	9.63 %	75.08 %
0.03 %	807	6,299	12.81 %	72.99 %
0.04 %	682	4,491	15.19 %	76.69 %
0.05 %	594	3,291	18.05 %	81.99 %
0.06 %	513	2,461	20.85 %	83.82 %
0.07 %	440	1,905	23.10 %	88.64 %
0.08 %	380	1,480	25.68 %	87.89 %
0.09 %	332	1,206	27.53 %	93.07 %
0.10 %	298	938	31.77 %	84.23 %

(b) Varying database size

DB size	#MaxHUI	#HUI	PRR	Prec.
50,000	594	3,291	18.05 %	81.99 %
100,000	601	3,376	17.80 %	95.51 %
150,000	649	3,011	21.55 %	78.89 %
200,000	620	3,420	18.13 %	91.13 %
250,000	664	3,586	18.52 %	89.46 %
300,000	631	3,254	19.39 %	84.31 %
350,000	613	3,192	19.20 %	87.11 %
400,000	591	3,472	17.02 %	92.22 %
450,000	631	3,090	20.42 %	77.18 %
500,000	646	3,264	19.79 %	83.75 %

6 Conclusions

In this paper, we proposed a novel framework, namely GUIDE, for efficiently mining maximal high utility itemsets from data streams. The methods for different models landmark, sliding window and time fading are proposed. The proposed compact data structure MUI-Trees are cooperated with the methods for storing essential information in data streams. Besides, two effective and efficient strategies are proposed for tracing and pruning the MUI-Trees. Main contributions of this work are listed as follows: 1) This work first addresses the problem of discovering compact forms of high utility itemsets from data streams; 2) GUIDE is an effective one-pass framework which meets the requirements of data stream mining; 3) MUI-Tree maintains essential information for the mining processes and the proposed strategies further facilitates the performance of GUIDE. 4) GUIDE generates compact and insightful patterns which are not only high utility but also maximal from the data streams. The experimental results show that GUIDE outperforms the compared algorithms substantially under the tested conditions. Generally speaking, GUIDE is an efficient and effective algorithm that can achieve the precision over 80% in average with performance improvement of saving over 90% of runtime .

Acknowledgement

This research was supported by National Science Council, Taiwan, R.O.C. under grant no. NSC100-2631-H-006-002 and NSC100-2218-E-006-017.

References

1. R. Agrawal and R. Srikant, “Fast Algorithms for Mining Association Rules,” in Proceedings of the 20th International Conference on Very Large Data Bases, pp. 487-499, September 1994.
2. C. F. Ahmed, S. K. Tanbeer, B.-S. Jeong, and Y.-K. Lee, “Efficient Tree Structures for High Utility Pattern Mining in Incremental Databases,” in IEEE Transactions on Knowledge and Data Engineering, Vol. 21, Issue 12, pp. 1708-1721, 2009.
3. A. Bifet, G. Holmes, B. Pfahringer, and R. Gavaldà, “Mining Frequent Closed Graphs on Evolving Data Streams,” in Proc. of the 17th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2011), pp. 591-599, San Diego, CA, USA, August, 2011.
4. R. Chan, Q. Yang and Y. Shen. “Mining high utility itemsets,” in Proc. of Third IEEE Int'l Conf. on Data Mining, pp. 19-26, Nov., 2003.
5. J. Cheng, Y. Ke and W. Ng, “Maintaining Frequent Closed Itemsets over a Sliding Window,” in Journal of Intelligent Information Systems (JIIS), Vol. 31, Issue 3, pp. 191-215, 2007.
6. J. Cheng, Y. Ke and W. Ng, “A survey on algorithms for mining frequent itemsets over data streams,” in Knowledge and Information Systems, Vol. 16, Issue 1, pp. 1-27, 2008.

7. Y. Chi, H. Wang, P. S. Yu and R. R. Muntz, “Moment: Maintaining Closed Frequent Itemsets over a Stream Sliding Window,” in Proc. of IEEE Int’l Conf. on Data Mining, pp. 59-66, 2004.
8. M. M. Gaber, A. B. Zaslavsky and S. Krishnaswamy, “Mining Data Streams: A Review,” in SIGMOD Record Vol. 34, No. 2, pp. 18-26, 2005.
9. C. Giannella, J. Han, J. Pei, X. Yan and P. S. Yu, “Mining Frequent Patterns in Data Streams as Multiple Time Granularities,” in H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha (eds.), Next Generation Data Mining, AAAI/MIT, pp. 191-212, 2003.
10. L. Golab and M. T. Ozsu, “Issues in Data Stream Management,” in ACM SIGMOD Record, Vol. 32, No. 2, pp. 5-14, June 2003.
11. K. Gouda and M. J. Zaki. “Efficiently Mining Maximal Frequent Itemsets,” in Proc. of the IEEE International Conference on Data Mining (ICDM), pp. 163-170, San Jose, 2001.
12. J. Han, J. Pei, and Y. Yin. “Mining Frequent Patterns without Candidate Generation”. In Proc. of ACM-SIGMOD Int’l Conf. on Management of Data (SIGMOD’00), pp. 1-12, May 2000.
13. N. Jiang and L. Gruenwald, “CFI-Stream: Mining Closed Frequent Itemsets in Data Streams,” in Proc. of the Utility-Based Data Mining Workshop, ACM KDD (UBDM’06), pp. 592-597, USA, August 2006.
14. N. Jiang and L. Gruenwald, “Research Issues in Data Stream Association Rule Mining,” in ACM SIGMOD Record, Vol. 35, No. 1, pp. 14-19, March 2006.
15. C. Jin, W. Qian, C. Sha, J. X. Yu and A. Zhou, “Dynamically Maintaining Frequent Items Over a Data Stream,” in Proc. of CIKM’03, pp. 287-294, Nov. 2003.
16. D. Lee and W. Lee, “Finding Maximal Frequent Itemsets over Online Data Streams Adaptively,” in Proc. of Fifth IEEE Int’l Conf. on Data Mining (ICDM’05), November, 2005.
17. C. K.-S. Leung and Q. I. Khan, “DSTree: A Tree Structure for the Mining of Frequent Sets from Data Streams,” in Proc. of the 6th IEEE Int'l Conf. on Data Mining (ICDM’06), pp. 928-932.
18. C. K.-S. Leung and F. Jiang, “Frequent Itemset Mining of Uncertain Data Streams using the Damped Window Model,” in Proc. of the 26th Annual ACM Symposium on Applied Computing, pp. 950-955, Taichung, Taiwan, March, 2011.
19. H. F. Li, C. C. Hob and S. Y. Lee, “Incremental Updates of Closed Frequent Itemsets over Continuous Data Streams,” in Expert Systems with Applications (ESWA) Vol. 36, Issue 2, pp. 2451-2458, 2009.
20. H. F. Li, H. Y. Huang, Y. C. Chen, Y. J. Liu, and S. Y. Lee, “Fast and Memory Efficient Mining of High Utility Itemsets in Data Streams,” in Proc. of the 8th IEEE Int'l Conf. on Data Mining (ICDM’08), pp. 881-886, 2008.
21. Y.-C. Li, J.-S. Yeh, and C.-C. Chang, “Isolated items discarding strategy for discovering high utility itemsets,” Data & Knowledge Engineering, Vol. 64, Issue 1, pp. 198-217, Jan. 2008.
22. C. H. Lin, D. Y. Chiu, Y. H. Wu and A. L. P. Chen, “Mining Frequent Itemsets from Data Streams with a Time-Sensitive Sliding Window,” in Proc. of SIAM Int’l Conf. on Data Mining, 2005.
23. Y. Liu, W. Liao and A. Choudhary, “A Fast High Utility Itemsets Mining Algorithm,” in Proc. of the Utility-Based Data Mining Workshop, pp. 90-99, August 2005.
24. X. Liu, J. Guan and P. Hu, “Mining Frequent Closed Itemsets from a Landmark Window over Online Data Stream,” in Computers & Mathematics with Applications, Vol. 57, Issue 6, pp. 927-936, 2009.
25. J. Pei, J. Han, and R. Mao, “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 (DMKD 2000), pp. 11-20, 2000.
26. B.-E. Shie, V. S. Tseng, and P. S. Yu, “Online mining of temporal maximal utility itemsets

from data streams," in Proc. of the 25th Annual ACM Symposium on Applied Computing, pp. 1622-1626, Switzerland, March, 2010.
27. V. S. Tseng, C. J. Chu and T. Liang, "Efficient Mining of Temporal High Utility Itemsets from Data streams," in Proc. of ACM KDD Workshop on Utility-Based Data Mining Workshop (UBDM'06), pp. 18-27, USA, August 2006.
28. V. S. Tseng, C. W. Wu, B. E. Shie, and P. S. Yu, "UP-Growth: An Efficient Algorithm for High Utility Itemsets Mining," in Proc. of the 16th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2010), pp. 253-262, Washington, DC, USA, July, 2010.
29. H. Yao, H. J. Hamilton and L. Geng, "A unified framework for utility-based measures for mining itemsets," in Proc. of ACM SIGKDD 2nd Workshop on Utility-Based Data Mining, pp. 28-37, USA, Aug., 2006.
30. S. J. Yen, Y. S. Lee, C. W. Wu, and C. L. Lin, "An Efficient Algorithm for Maintaining Frequent Closed Itemsets over Data Stream," in Proc. of IEA/AIE 2009, LNAI 5579, pp. 767–776, Tainan, Taiwan, 2009.
31. S. J. Yen, C. W. Wu, Y. S. Lee and V. S. Tseng, "A Fast Algorithm for Mining Frequent Closed Itemsets over Stream Sliding Window," in Proc. of IEEE Int'l Conf. on Fuzzy Systems (FUZZ-IEEE'2011), pp. 996-1002, Taipei, Taiwan, 2011.
32. Frequent Itemset Mining Implementations Repository, http://fimi.cs.helsinki.fi/