



Dynamic maintenance model for high average-utility pattern mining with deletion operation

Jimmy Ming-Tai Wu¹ · Qian Teng¹ · Shahab Tayeb² · Jerry Chun-Wei Lin³

Accepted: 18 May 2021
© The Author(s) 2021

Abstract

The high average-utility itemset mining (HAUIM) was established to provide a fair measure instead of genetic high-utility itemset mining (HUIM) for revealing the satisfied and interesting patterns. In practical applications, the database is dynamically changed when insertion/deletion operations are performed on databases. Several works were designed to handle the insertion process but fewer studies focused on processing the deletion process for knowledge maintenance. In this paper, we then develop a PRE-HAUI-DEL algorithm that utilizes the pre-large concept on HAUIM for handling transaction deletion in the dynamic databases. The pre-large concept is served as the buffer on HAUIM that reduces the number of database scans while the database is updated particularly in transaction deletion. Two upper-bound values are also established here to reduce the unpromising candidates early which can speed up the computational cost. From the experimental results, the designed PRE-HAUI-DEL algorithm is well performed compared to the Apriori-like model in terms of runtime, memory, and scalability in dynamic databases.

Keywords Pre-large · High average-utility itemset · Dynamic database · Transaction deletion

1 Introduction

Knowledge Discovery in Database (KDD) [2, 5, 8, 19, 32] is a way to obtain useful and interesting patterns/information from the databases. At present, KDD has been widely used in many domains and applications. Association-rule mining (ARM) [1] is regarded as the generic and basic research in KDD for identifying frequent itemsets (FIs) by minimum support threshold and then discovering association rules (ARs) by minimum confidence threshold. Apriori algorithm [1] was firstly proposed to discover association rules (ARs), which utilizes the generate-and-test model for identifying the possible candidates level by level. In order to improve the mining efficiency, Han et al. [9] built a frequent pattern (FP)-tree structure to compress the database by keeping

necessary information and process an efficient FP-growth mining algorithm to reveal the set of frequent itemsets (FIs). Then, a new structure called N-list was introduced by Deng et al. [5] and PrePost algorithm was presented to find frequent itemsets (FIs) efficiently. In order to effectively discover different types of patterns, many approaches have presented in different fields and applications [8, 19, 32].

However, in traditional ARM, it is assumed that each item is purchased at most once in each transaction and only the amount of appearance of items in databases is considered without other meaningful factors, i.e., significance, weight, or unit profit. A frequent item/itemset can bring frequency information to retailers, but cannot guarantee high profits for retailers. High-utility itemset mining (HUIM) [7, 22, 23, 28] was provided by looking at both quantity and unit of profit of items to reveal high-utility itemsets (HUIs). An itemset is considered as a high-utility itemset (HUI) if its measured utility is no less than a pre-defined minimum utility threshold. Regarding the downward closure (DC) property, the initial HUIM does not hold DC thus the “combinational explosion” happens in the mining process. Transaction-weighted utilization (TWU) model [23] was defined by Liu et al. that designs an over-estimated upper-bound utility on an itemset. Thus, the size of the unpromising candidates can be greatly reduced.

This article belongs to the Topical Collection: *Emerging topics in Applied Intelligence selected from IEA/AIE2020*
Guest Editors: Hamido Fujita, Philippe Fournier-Viger and Moonis Ali

✉ Jerry Chun-Wei Lin
jerrylin@ieee.org

Extended author information available on the last page of the article.

Nevertheless, TWU model follows the generate-and-test process, in which a large number of profitable HUIs can thus be generated. To address this limitation, high-utility pattern (HUP)-tree [16] was introduced to keep the promising 1-itemsets into the tree structure. Tseng et al. [25] further introduced the UP-growth+ approach based on the UP-tree structure to reveal the set of HUIs accurately. Moreover, utility-list (UL) structure [22] was proposed to obtain the k -itemsets based on the simple join operation. Many approaches have been respectively presented and discussed and most of them were used in the static applications [21, 26, 29, 33]. Moreover, although the HUIM can discover high-utility itemsets in practical applications, the total profit value of an itemset increases along with its length. For instance, diamonds combine with any commodities would generate high utility and this combination might be regarded as a HUI. To evaluate the high utility patterns fairly, high average-utility itemset mining (HAUIM) [11] concerns the number of items in an itemset to reveal the set of high average-utility itemsets (HAUIs). If an itemset is called as a HAUI, the average-utility of this itemset exceeds the user-designed minimum threshold of average-utility. Hong et al. [11] proposed a level-wise manner called TPAU to reveal HAUIs, and this method introduced “average-utility upper-bound (*auub*)” to retain the DC property. This *auub* model is used to find high average-utility upper bound itemsets (HAUUBIs), that the average-utility of a HAUUBI is larger than the defined threshold value and the set of HAUUBIs also keeps the completeness and correctness of the satisfied HAUIs. To increase mining efficiency, a novel structure called high average-utility pattern (HAUP)-tree was designed by Lin et al. [15] to maintain 1-HAUUBIs. Lin et al. [18] further designed average-utility list (AUL)-structure to retrieve the k -itemsets quickly based on the simple join operation. Other extensions of HAUIM [12, 17] were also discussed and studied to improve the efficiency problem of knowledge discovery.

The above existing approaches related to HAUIM focused on mining HAUIs in the static database, but in practical applications, the content of a database is dynamically changed according to different operations, i.e., insertion or deletion. In this case, traditional methods to handle this situation is to re-perform the mining progress again to re-scan the updated database that causes an additional calculation. Cheung et al. developed Fast UPdated (FUP) [3] concept and FUP2 [4] concept to retain the discovered itemsets when some transactions are inserted or deleted, respectively. However, HAUIM, which applies the FUP concept, must also perform re-scan process for mining the required information from the updated transactions, which is very costly. Other extensions of dynamic database [13, 27, 31] were also discussed and studied to improve the efficiency of mining process. In this

paper, we first take advantage of pre-large concept [10] for updating the discovered HAUIs when some transactions are removed in the dynamic databases. Contributions of this paper are shown as follows.

1. A PRE-HAUI-DEL algorithm based on pre-large technique is first introduced for maintaining and updating the discovered HAUIs for transaction deletion.
2. A safety bound is defined (for pre-large itemsets) to reduce the multiple database scans while transactions are removed. Scanning the updated database only happens while there the accumulated number of transactions is then deleted. This strategy, in turn, significantly improves the maintenance performance.
3. Two upper-bound values are then established here to reduce the unpromising candidates in the early stage, thus the performance of the designed model is retained.
4. Numerous experiments are then implemented to verify the performance of the designed PRE-HAUI-DEL algorithm compared to the Apriori-like approach in terms of runtime, memory usage, and the scalability.

2 Review of related work

High average-utility itemset mining (HAUIM) and the concept of pre-large are separately introduced in this section.

2.1 High average-utility itemset mining

The fundamental algorithms of ARM [1] are designed to find patterns by only considering the frequency of itemsets. However, these algorithms can simply manage the binary database but the other interesting factors such as the importance, weight, or unit profit and quantity are ignored. To discover more sufficient and significant information, HUIM (high-utility itemset mining) [6, 23, 28] concerns both quantity and unit profit of items to obtain the profitable patterns from databases. In order to maintain the downward closure (DC) property for the mining progress, transaction-weighted utilization (TWU) model [23] was applied to keep the DC property of the high transaction-weighted utilization itemsets (HTWUIs). Lin et al. [16] further constructed high-utility pattern (HUP)-tree to store the 1-HTWUIs in a tree structure, which can improve the mining efficiency in HUIM. To apply HUP-tree structure, HUP-tree algorithm was designed to reveal the set of HUIs efficiently. Compared with the traditional Apriori-like method, HUP-tree approach is more efficient. To better enhance the mining performance, Tseng et al. [25] developed the UP-tree structure to reserve essential information for mining process, and designed UP-Growth+ algorithm and several pruning strategies for

finding HUIs. Liu et al. [22] then proposed a utility-list (UL) structure to quickly obtain the k -candidate HUIs by a very simple join operation. Based on the UL structure, the HUI-Miner algorithm was designed to reveal HUIs efficiently. Many studies of HUIM [20–22, 26, 29, 33] have been extensively discussed and developed to improve the mining process in KDD.

Although HUIM has the ability to discover profitable information for decision-making, the utility value increases along with the size of the itemset. Due to this consequence, it might produce the misunderstanding information for decision-making. For example, any combinations with caviar/diamond may be considered as a HUI as well. To evaluate the high utility patterns fairly, high average-utility itemset mining (HAUIM) [11, 15] was presented to take into account the length of itemsets for revealing high average-utility itemsets (HAUIs). The first algorithm related to HAUIM is called TPAU [11], which uses average-utility upper-bound (*auub*) model to keep downward closure (DC) property in a Apriori-like process. Lin et al. [15] then developed HAUP-tree, which is a compressed tree structure to maintain the 1-itemsets, thus the search space can be greatly reduced. HAUI-Miner algorithm applied a link-list structure to store crucial information based on a novel list structure. Several approaches for HAUIM [14, 17, 24, 30] have been studied and the development of HAUIM is still in progress.

2.2 Pre-large concept

Traditional pattern mining only considered to find the significant information from a static dataset, such as association-rule mining (ARM) [1], frequent itemset mining (FIM) [2], and high-utility itemset mining (HUIM) [6]. When a database is dynamically changed, for instance, transaction deletion, insertion or modification, the batch approaches have to examine the updated database again to maintain the discovered knowledge. Thus, the previous calculations and results have become invalid and the additional cost is necessary and required.

To deal with the dynamic data mining efficiently, Cheung et al. then developed Fast UPdated (FUP) [3] concept and FUP2 [4] concept to keep the revealed itemsets when the size of the dataset is changed. For instance of transaction deletion in FUP2 concept, it separates the discovered patterns into four cases according to the original dataset and the deleted one, and each case has its own designed model to maintain and update the discovered information. However in some cases, the original database is still required to be rescanned for updating the discovered patterns, thus it takes some computational cost for knowledge maintenance.

To maintain the discovered information efficiently and decrease the number of database rescans, Hong

et al. [10] furtherer proposed the pre-large concept for the maintenance progress. Based on the pre-large concept, two thresholds were designed that keeps not only the large itemsets but also the pre-large itemsets for knowledge maintenance. The pre-large itemsets can thus be considered as a buffer to avoid the extra movements of an itemset from large (high) to small and vice-versa. Based on the benefits of pre-large concept, the re-scanning process of databases can only be performed while a certain amount of transactions are inserted into the databases or deleted from the databases. This idea can greatly reduce the computational cost of multiple database scans. In pre-large concept for transaction deletion, nine cases are then arisen and shown in Fig. 1.

In Fig. 1, for the cases 2 to 4, 7, and 8, the final results are the same as the results before transaction deletion. However, the number of discovered knowledge may be reduced in case 1, and some new patterns can thus be produced in cases 5, 6, and 9. If the large itemsets and the pre-large itemsets are maintained and stored from the original database, it is simple to maintain cases 1, 5, and 6. For case 9, if the number of deleted transactions is no large than the safety bound (f), the itemsets in case 9 are impossible to be a large itemset in the updated database. The safety bound (f) is defined as:

$$f = \lfloor \frac{(S_u - S_l) \times TU^D}{S_u} \rfloor, \tag{1}$$

where S_u and S_l are respectively the upper-bound threshold and the lower bound threshold, which can be both defined by user or an expert. The possible results of the all cases are given in Table 1.

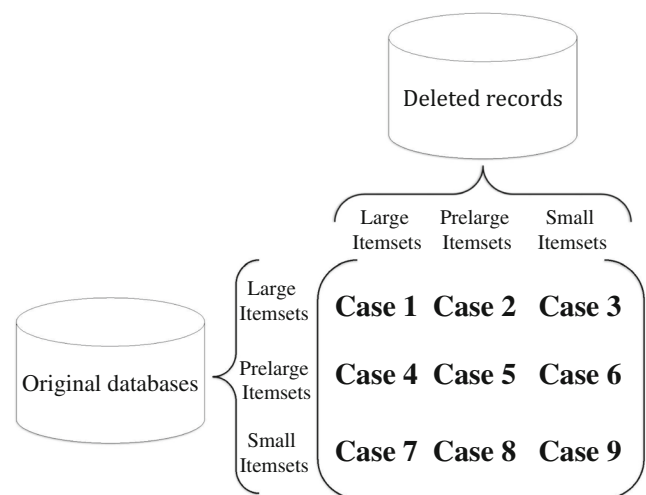


Fig. 1 Nine cases of the pre-large concept

Table 1 Nine cases regarding L (large), PL (pre-large) and S (small) for deletion

Case	Original/Deleted DB	Possible Results
1	L/L	L, PL, S
2	PL/L	L
3	L/S	L
4	PL/L	PL, S
5	PL/PL	L, PL, S
6	PL/S	L, PL
7	S/L	S
8	S/PL	S
9	S/S	PL, S

3 Definitions and problem statement

3.1 Partial upper-bound and high partial upper-bound itemset

In general, *auub* property has been widely used in the past methods of HAUIM. The average utility of an itemset in HAUIM is smaller than *auub* value since the *auub* is an over-estimated value in HAUIM that is used to keep the DC property in the mining progress. However, *auub* is too large thus there will be many unpromising candidates generated. To solve this limitation, the proposed *pub* was designed and defined as below.

Definition 1 An item is an active item (*ai*) if this item at least exists in one of *pubis*. The set of *ai* is denoted as *ais_n* and the formal definition is given below as:

$$ais_n = \{ai \in x \mid x \in pubis^{n-1}\}, \tag{2}$$

where *pubisⁿ⁻¹* is a *pubis* length is set as (*n* - 1).

Definition 2 *rmua(i, t)* is the remaining maximal utility of an item *i* in a transaction *t* and the formal definition is given below as:

$$rmua(i, t) = \max\{u(i) \mid i \in (ais \cap I_t) \setminus i\}, \tag{3}$$

where *I_t* is a set of the purchase items in this transaction *t*.

Definition 3 *pub_i^t* is the partial upper-bound of an item *i* in a transaction *t* and the formal definition is given below as:

$$pub_i^t = \begin{cases} \frac{u(i,t)+m \times rmua(i,t)}{|i|+m}, & rmua(i, t) > au(i, t) \\ \frac{u(i,t)+rmua(i,t)}{|i|+1}, & 0 < rmua(i, t) \leq au(i, t) \\ 0, & rmua(i, t) = 0, \end{cases} \tag{4}$$

where *m* is the length of (*ais* ∩ *I_t*) \ *i*.

Definition 4 *pub_i* is the partial upper-bound of an item *i* and the formal definition is given below as:

$$pub_i = \sum_{t \in D} pub_i^t. \tag{5}$$

Definition 5 An itemset *X* is a high partial upper-bound itemset (*hpubi*) if its *pub* is not less than a predefined threshold *r*, which indicates that *pub_i ≥ r*.

3.2 Lead partial upper-bound and high lead partial upper-bound itemset

In this section, a smaller itemset is recommended to prevent the further generation of candidate sets. This itemset is called the high lead partial upper-bound itemset (*hlpubi*), and it is a subset of *hpubi*. The definitions of *lpub* and *hlpubi* are given below.

The predefined item order is defined as *L* = {*i₁^l, i₂^l, ..., i_p^l*}. Assume that *i_j = i_w^l*, and set an itemset *s* = {*i_{w+1}^l, i_{w+2}^l, ..., i_p^l*}.

Definition 6 *lrmua(i, t)* is the lead remaining maximal utility of an item *i* in a transaction *t* and the formal definition is given below as:

$$lrmua(i, t) = \max\{u(i) \mid i \in (ais \cap I_t \cap s) \setminus i\}. \tag{6}$$

Definition 7 *lpub_i^t* is the lead partial upper-bound of an item *i* in a transaction *t* and the formal definition is given below as:

$$lpub_i^t = \begin{cases} \frac{u(i,t)+m \times rmua(i,t)}{|i|+m}, & lrmua(i, t) > au(i, t) \\ \frac{u(i,t)+rmua(i,t)}{|i|+1}, & 0 < lrmua(i, t) \leq au(i, t) \\ 0, & lrmua(i, t) = 0, \end{cases} \tag{7}$$

where *m* is the length of (*ais* ∩ *I_t* ∩ *s*) \ *i*.

Definition 8 *lpub_i* is the lead partial upper-bound of an item *i* and the formal definition is given below as:

$$lpub_i = \sum_{t \in D} lpub_i^t. \tag{8}$$

Definition 9 An itemset *X* is a high lead partial upper-bound itemset (*hlpubi*) if its *lpub* is not less than a predefined threshold *r*, which indicates that *lpub_i ≥ r*.

3.3 Problem statement

In dynamic situation of transaction deletion based on the pre-large concept, there are nine cases that should be considered and arisen. Assume *d* is the deleted transactions, and |*d*| is the number of transactions in *d*. The problem

statement of the HAUIM for transaction deletion is defined as follows:

Problem Statement For the HAUIM with transaction deletion, an efficient model is required for development that can efficiently maintain the discovered knowledge without multiple database scans while the transactions are deleted from the databases. Thus, a satisfied itemset in the database for transaction deletion is considered as an HAUIM if it satisfies the condition as:

$$HAUIM \leftarrow \{X \mid au(X)^U \geq (TU^D - TU^d) \times S_u\}, \quad (9)$$

where $au(X)^U$ is the average-utility of itemset X in the updated database, TU^D and TU^d are respectively the total utility in D and d .

4 Proofs of nine cases

In this part, the proofs of nine cases of the proposed method based on the pre-large concept are given below.

Lemma 1 For case 1, an itemset X is large in both the original database D ($auub(X)^D \geq TU^D \times S_u$) and the deleted database d ($auub(X)^d \geq TU^d \times S_u$), then it will be large, pre-large, or small in the updated database U .

Proof For $auub(X)^D \geq TU^D \times S_u$ and $auub(X)^d \geq TU^d \times S_u$, if $auub(X)^D - auub(X)^d \geq TU^D \times S_u - TU^d \times S_u$ then X is large in the updated database U ; if $auub(X)^D - auub(X)^d < TU^D \times S_u - TU^d \times S_u$ then X is pre-large in the updated database U ; if $auub(X)^D - auub(X)^d \leq TU^D \times S_l - TU^d \times S_l$ then X is small in the updated database U . \square

Lemma 2 For case 2, an itemset X is large in the original database D ($auub(X)^D \geq TU^D \times S_u$) and pre-large in the deleted database d ($TU^d \times S_l \leq auub(X)^d < TU^d \times S_u$), then it will be large in the updated database U .

Proof For $auub(X)^D \geq TU^D \times S_u$ and $TU^d \times S_l \leq auub(X)^d < TU^d \times S_u$, then $auub(X)^D - auub(X)^d \geq TU^D \times S_u - TU^d \times S_u$, even $auub(X)^U \geq TU^U \times S_u$, so X is large in the updated database U . \square

Lemma 3 For case 3, an itemset X is large in the original database D ($auub(X)^D \geq TU^D \times S_u$) and small in the deleted database d ($auub(X)^d < TU^d \times S_l$), then it will be large in the updated database U .

Proof For $auub(X)^D \geq TU^D \times S_u$, $auub(X)^d < TU^d \times S_l$ and $S_u > S_l$, then $auub(X)^D - auub(X)^d > TU^D \times S_u - TU^d \times S_u$, even $auub(X)^U \geq TU^U \times S_u$, so X is large in the updated database U . \square

Lemma 4 For case 4, an itemset X is pre-large in the original database D ($TU^D \times S_l \leq auub(X)^D < TU^D \times S_u$) and large in the deleted database d ($auub(X)^d \geq TU^d \times S_u$), then it will be pre-large or small in the updated database U .

Proof For $TU^D \times S_l \leq auub(X)^D < TU^D \times S_u$ and $auub(X)^d \geq TU^d \times S_u$, then $auub(X)^D - auub(X)^d < TU^D \times S_u - TU^d \times S_u$, even $auub(X)^U < TU^U \times S_u$, so X will be pre-large and small in the updated database U . \square

Lemma 5 For case 5, an itemset X is pre-large in both the original database D ($TU^D \times S_l \leq auub(X)^D < TU^D \times S_u$) and the deleted database d ($TU^d \times S_l \leq auub(X)^d < TU^d \times S_u$), then it will be large, pre-large, or small in the updated database U .

Proof For $TU^D \times S_l \leq auub(X)^D < TU^D \times S_u$ and $TU^d \times S_l \leq auub(X)^d < TU^d \times S_u$, if $auub(X)^D - auub(X)^d \geq TU^D \times S_u - TU^d \times S_u$ then X is large in the updated database U ; if $auub(X)^D - auub(X)^d < TU^D \times S_u - TU^d \times S_u$ then X is pre-large in the updated database U ; if $auub(X)^D - auub(X)^d \leq TU^D \times S_l - TU^d \times S_l$ then X is small in the updated database U . \square

Lemma 6 For case 6, an itemset X is pre-large in the original database D ($TU^D \times S_l \leq auub(X)^D < TU^D \times S_u$) and small in the deleted database d ($auub(X)^d < TU^d \times S_l$), then it will be large or pre-large in the updated database U .

Proof For $TU^D \times S_l \leq auub(X)^D < TU^D \times S_u$ and $auub(X)^d < TU^d \times S_l$, then $auub(X)^D - auub(X)^d \geq TU^D \times S_l - TU^d \times S_l$, even $auub(X)^U \geq TU^U \times S_l$, so X will be large and pre-large in the updated database U . \square

Lemma 7 For case 7, an itemset X is small in the original database D ($auub(X)^D < TU^D \times S_l$) and large in the deleted database d ($auub(X)^d \geq TU^d \times S_u$), then it will be small in the updated database U .

Proof For $auub(X)^D < TU^D \times S_l$, $auub(X)^d \geq TU^d \times S_u$ and $S_u > S_l$, then $auub(X)^D - auub(X)^d < TU^D \times S_l - TU^d \times S_l$, even $auub(X)^U < TU^U \times S_l$, so X is small in the updated database U . \square

Lemma 8 For case 8, an itemset X is small in the original database D ($auub(X)^D < TU^D \times S_l$) and pre-large in the deleted database d ($TU^d \times S_l \leq auub(X)^d < TU^d \times S_u$), then it will be small in the updated database U .

Proof For $auub(X)^D < TU^D \times S_l$ and $TU^d \times S_l \leq auub(X)^d < TU^d \times S_u$, then $auub(X)^D - auub(X)^d < TU^D \times S_l - TU^d \times S_l$, even $auub(X)^U < TU^U \times S_l$, so X is small in the updated database U . \square

Lemma 9 For case 9, an itemset X is small in both the original database D ($auub(X)^D < TU^D \times S_l$) and the deleted database d ($auub(X)^d < TU^d \times S_l$), then it will be pre-large or small in the updated database U .

Proof For $auub(X)^D < TU^D \times S_l$ and $auub(X)^d < TU^d \times S_l$, if $auub(X)^D - auub(X)^d \geq TU^D \times S_u - TU^d \times S_u$ then X is large in the updated database U ; if $auub(X)^D - auub(X)^d < TU^D \times S_u - TU^d \times S_u$ then X is pre-large in the updated database U ; if $auub(X)^D - auub(X)^d \leq TU^D \times S_l - TU^d \times S_l$ then X is small in the updated database U . \square

Theorem 1 X is a small itemset (not a high average-utility itemset nor pre-large itemset) in the original database and the deleted database. It is impossible to be a high average-utility itemset in the updated database if the total utility of the deleted transactions is not larger than the following safety bound as:

$$TU^d < f = \lfloor \frac{(S_u - S_l) \times TU^D}{S_u} \rfloor. \quad (10)$$

Here, theorem 1 will be used in the proposed algorithm in which the pre-large itemsets serve as a buffer to store some extra information. It is thus sufficient to reduce the time by performing the multiple database scans and further diminish the updating cost of the deletion operation.

5 Proposed pre-large-based framework for transaction deletion

An innovative algorithm called PRE-HAUI-DEL is proposed to maintain and update the set of HAUIs in a dynamic database efficiently, and this algorithm is based on Apriori-like method. This section introduces the detailed Apriori framework for maintaining and updating the set of HAUI in dynamic databases. In the process of knowledge updating, the PRE-HAUI-DEL algorithm uses the novel upper-bound value to find new itemsets, thus pruning the optimistic

candidates early and reducing the search space. First, a novel partial upper-bound (*pub*) is suggested to prune the unpromising candidates, and an itemset is named as a high partial upper-bound itemset (*hpubi*) if its utility exceeds the *pub*. Second, there will be a subset selected from *hpubi*, called *hlpubi* (high lead-*pubi*). In fact, these two sets are produced simultaneously, but it uses a tighter upper-bound than *pub* called lead-*pub* (*lpub*). The smallest upper-bound can further prevent the generation of candidate sets.

5.1 The designed framework

As shown in Fig. 2, the algorithm first mines the HAUI, pre-large itemsets and *pubis*, *lead-pubis* from the original database, If some data is deleted, then calculate the threshold r . According to the size of r , the next step is to re-scan the updated database again or update the new HAUI from the obtained itemsets. If you want to continue to delete the dataset, update the threshold r , and proceed to the next round.

5.2 Proposed PRE-HAUI-DEL algorithm

To efficiently update the discovered information for transactions deletion, a novel algorithm called PRE-HAUI-DEL is developed. The proposed algorithm is shown in Algorithms 1 and 2. For Algorithm 1, it is an Apriori-based process to discover the sets of high average-utility itemsets and pre-large itemsets from the original database. Moreover, it utilizes two upper-bounds (*pub* and *lpub*) to generate the candidate itemsets.

First in Algorithm 1, the proposed method calculates the safety-bound value for database rescan in **line 2** and scans the whole original database for each item to reveal the set of HAUIs with one item and initialize *pubis* and *lead-pubis* in **lines 3 to 15**. Note that the proposed process uses *auub* to initialize *pubis* and *lead-pubis* in **line 12**. Then a while loop in **lines 16 to 38** is performed to evaluate the candidate itemsets and reveal HAUIs. This while loop first generates the candidate itemsets by performing a sub-loop for *lead-pubis* in **lines 18 to 20**. The first $n-1$ itemsets of each candidate itemset with n items should be included in *lead-pubis*. For instance, if the itemsets “{ a, b, c }” are the candidate itemset, the itemset “{ a, b }” definitely exists in *lead-pubis* and the itemsets “{ a, c }” and “{ b, c }” absolutely exists in *pubis*. In fact, according to the above definitions, “{ a, b }” are also included in *pubis* since *lead-pubis* is a smaller set than *pubis*, it can reduce the searching space for the generation progress of the candidate itemsets. The second loop here is to evaluate the whole candidate itemsets

into the sets of *pubi*, *lead-pubi*, pre-large itemset and high average-utility itemset by checking their *pub*, *lpub* and *au*, respectively. The sets of *pubi* and *lead-pubi* are used to generate the candidate itemsets for the next round and the sets of pre-large itemset and high average-utility itemset will be used in Algorithm 2 to update the final result with the safety-bound value *r*.

Algorithm 1 PRE-HAUI-DEL.

Input: *D*, original transaction dataset,
 S_u , upper-bound utility threshold,
 S_l , lower-bound utility threshold.
Output: *P*, set of pre-large itemsets,
H, set of HAUI.

- 1: initialize $H = \emptyset, P = \emptyset$;
- 2: calculate the threshold *r* for *D* to perform re-scan process;
- 3: **for** each transaction *t* in *D* **do**
- 4: **for** each item *i* in *t* **do**
- 5: calculate *auub*(*i*) and *au*(*i*);
- 6: **if** $au(i) \geq S_u \times TU^D$ **then**
- 7: put *i* in *H*;
- 8: **else if** $au(i) \geq S_l \times TU^D$ **then**
- 9: put *i* in *P*;
- 10: **end if**
- 11: **if** $auub(i) \geq S_l \times TU^D$ **then**
- 12: put *i* in *pubis* and *lead-pubis*
- 13: **end if**
- 14: **end for**
- 15: **end for**
- 16: **while** *pubis* $\neq \emptyset$ and *lead-pubis* $\neq \emptyset$ **do**
- 17: initialize *C* = \emptyset ;
- 18: **for** each itemset *e* in *lead-pubis* **do**
- 19: generate candidate itemsets by combining *e* and the itemsets in *pubis* $\rightarrow C$
- 20: **end for**
- 21: initialize *pubis* $\leftarrow \emptyset, lead-pubis \leftarrow \emptyset$;
- 22: **for** each transaction *t* in *D* **do**
- 23: calculate all of necessary utility information for each itemset in *t*;
- 24: **end for**
- 25: **for** each candidate itemset *c* in *C* **do**
- 26: **if** $pub_c \geq S_l \times TU^D$ **then**
- 27: put *c* in *pubis*;
- 28: **end if**
- 29: **if** $lpub_c \geq S_l \times TU^D$ **then**
- 30: put *c* in *lead-pubis*;
- 31: **end if**
- 32: **if** $au(c) \geq S_u \times TU^D$ **then**
- 33: put *c* in *H*;
- 34: **else if** $au(c) \geq S_l \times TU^D$ **then**
- 35: put *c* in *P*;
- 36: **end if**
- 37: **end for**
- 38: **end while**
- 39: **return** *P, H, r*;

Algorithm 2 Maintenance process of PRE-HAUI-DEL.

Input: *d*, deleted transaction dataset,
P, pre-large itemsets,
H, high average-utility itemsets,
r, re-scan threshold,
 S_u , upper-bound utility threshold,
 S_l , lower-bound utility threshold.
Output: *P*, updated pre-large itemsets,
H, updated HAUI,
r, updated re-scan threshold.

- 1: set *s* is the size of *d*;
- 2: $r = r - s$;
- 3: **if** $r \geq 0$ **then**
- 4: perform the complete re-scanning process (Algorithm 1) to obtain *P'*, *H'*;
- 5: $S \leftarrow (P' \cup H') \cap (P \cup H)$;
- 6: calculate necessary utility information for *S* in *d*;
- 7: update the utility value of *P, H* from *P, H, P', H', S*
- 8: **else**
- 9: generate $D \leftarrow D - d$;
- 10: re-scan the updated dataset *D* by performing Algorithm 1;
- 11: update the utility value of *P, H, r*;
- 12: **end if**
- 13: **return** *P, H, r*;

In Algorithm 2, it is an updating process for the itemsets in a dynamic database with several deletion operations. By obtaining the benefit of pre-large concept, the pre-large itemsets come across as a buffer to retain the edge itemsets which are almost high average-utility itemsets. The re-scan threshold makes sure the other itemsets out of the pre-large itemset is impossible to become a high average-utility itemset if the number of deleted transactions does not achieve the safety-bound value. In line 2, the updating process first checks the size of the deleted transactions and updates the re-scan threshold. Suppose the size of the deleted transactions already achieves the threshold, in that case, the proposed method will generate the updated dataset and perform Algorithm 1 again to update the pre-large itemsets, high average-utility itemsets, and re-scan

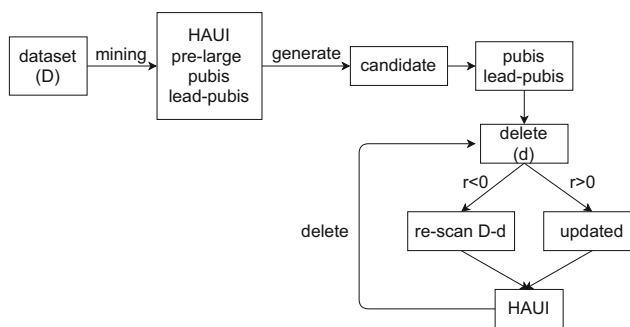


Fig. 2 The diagram of the designed framework

threshold r in **lines 4 to 7**. Otherwise, the proposed updating progress will utilize the pre-large itemsets and previous high average-utility itemsets to update discovered information, as well as updating the threshold value in **lines 9 to 11**. Finally, the proposed method will output the final P (pre-large itemsets), H (high average-utility itemsets) and re-scan threshold r as the final results in **line 13**.

6 An illustrated example

In this section, an example for maintaining and updating HAUIs from a dynamic database is given. Table 2 is used as the original database, and suppose the last 4 transactions (7 to 10) are then deleted from the database. In addition, Table 3 is the profit table, and we assume that the $S_u = 0.3$ and $S_l = 0.15$. The process of the designed algorithm is described below.

First, the designed algorithm calculate the safety-bound value as: $r = \lfloor \frac{(S_u - S_l) \times TU^D}{S_u} \rfloor = 280$ (Algorithm 1, **line 2**). After that, the designed algorithms generates 1-HAUIs, 1-HAUUBIs, 1-PAUUBIs, $pubis$ and $lead-pubis$ from original database (Algorithm 1, **lines 3 to 15**). The results of the $auub$ values for 1-itemsets in the original database are then shown in Table 4.

Here, we can obtain that $S_u \times TU^D = 168$ and $S_l \times TU^D = 84$. Thus, HAUIs = {C, D}, HAUUBIs = {B, C, D}, PAUUBIs = {E} and $pubis = lead-pubis = \{A, B, C, D, E\}$ can be obtained. After that, the algorithm calculates $s = TU^d = 210$, and updates $r = 280 - 210 = 90 > 0$ (Algorithm 2, **lines 1 to 2**). We then can also obtain the sets of 1-HAUIs, 1-HAUUBIs, 1-PAUUBIs, $pubis$ and $lead-pubis$ from the deleted database (Algorithm 2, **line 4**). Also, the $auub$ values for the deleted transactions of 1-itemsets are shown in Table 5.

The designed algorithm then calculate $S_u \times TU^d = 63$ and $S_l \times TU^d = 31.5$. In addition, HAUIs = {C, D},

Table 3 Profit table

Item	Profit
A	6
B	2
C	15
D	7
E	10

HAUUBIs = {B, C, D}, PAUUBIs = {E} and $pubis = lead-pubis = \{B, C, D, E\}$ are then obtained. The HAUUBIs and PAUUBIs are the updated (Algorithm 2, **lines 5 to 7**), and the results are respectively shown in Tables 6 and 7.

Since $TU^U = 350$, we then can obtain that $TU^U \times S_u = 105$. Thus, the final HAUIs is generated as: {C, E}. After that, the candidates from $pubis$ and $lead-pubis$ are then used to generate k -HAUIs (Algorithm 1, **lines 16 to 38**). After the recursive process of the designed algorithm based on the generate-and-test approach, we then can get the maintenance results as the output of the developed algorithm.

7 Experimental results

In this paper, two suggested upper-bound (pub and $lpub$) are further utilized to reduce the search space of candidate itemsets. This section describes two experimental processes of the proposed PRE-HAUI-DEL regarding two upper-bound models; one is with the $lpub$ and another is without the $lpub$, and the experiments showed that the proposed $lpub$ can reduce more candidate sets from being generated. The experimental results show the effectiveness of PRE-HAUI-DEL with different parameter settings on six different real datasets. The datasets applied in the experiments and their characteristics are given in Table 8. The experiments were executed on a Mac with Mojave operating system, which is equipped with Intel i5-5257U CPU and 8 GB main memory. The Java language is then used for the program implementation of the compared algorithms.

In the experiments, four algorithms are then compared and discussed, which are Apriori, Apriori ($lpub$), PRE-HAUI-DEL and PRE-HAUI-DEL ($lpub$). Apriori is the algorithm that preforms the re-scan process by Algorithm 1 but does not use $lpub$; PRE-HAUI-DEL is the algorithm that

Table 2 Original database

TID	A	B	C	D	E
1	6	0	0	0	0
2	0	1	0	5	0
3	0	6	0	0	0
4	0	0	5	0	0
5	6	0	0	0	0
6	6	0	0	0	0
7	6	0	0	0	0
8	6	0	0	0	0
9	6	0	0	0	0
10	6	0	0	0	0

Table 4 $auub$ values for original transactions of 1-itemsets

1-itemsets	A	B	C	D	E
$auub$	76	188	199	168	130
au	48	44	180	168	130

Table 5 *auub* values for deleted transactions of 1-itemsets

1-itemsets	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
<i>auub</i>	105	84	105	10
<i>au</i>	20	78	105	10

performs the updated process (Algorithm 2) but does not use *lpub*, Apriori (*lpub*) is the algorithm that re-scans the updated dataset when some transactions are deleted in the original dataset by applying Algorithm 1, and PRE-HAUI-DEL (*lpub*) is the algorithm that performs the updating process by Algorithm 2.

7.1 Runtime with different upper-bound utility threshold

The runtimes of the developed PRE-HAUI-DEL under different upper-bound utility thresholds in six real datasets are discussed in this section. Experimental results are given in Fig. 3.

From Fig. 3, it can be seen that whether the *lpub* is applied in the designed algorithm, the proposed PRE-HAUI-DEL is always better than the previous models. Without *lpub* strategy, the designed PRE-HAUI-DEL still well performs than the generic Apriori-like approaches. This reason is that the designed model does not require multiple database scans each time while the transactions are deleted from the database. However, Apriori (without pre-large technique) needs to scan the database even if there is a tiny amount of transactions deleted from the database. This computational cost is huge, especially when the database is frequently updated. Mathematically, the utilized *lpub* definitely is a tighter upper-bound than that of both *pub* or *auub*. However, the algorithm with *lpub* still suffers the computational cost in some cases, which can be seen in Fig. 3e while the threshold is set as 1.74%. The reason is even the number of reduced candidates is reduced, the designed algorithm requires to take the huge computational cost by holding the *lpub* property. In general, the *lpub* usually enhances the performance in most cases, and it can be seen from Fig. 3d that the Apriori with *lpub* sometimes can obtain better performance than the proposed PRE-HAUI-DEL without *lpub*.

Table 6 Updated *auub* 1-itemsets of HAUUBI in *D*

1-itemsets	<i>B</i>	<i>C</i>	<i>D</i>
<i>auub</i>	83	115	63
<i>au</i>	24	105	63

Table 7 Updated *auub* 1-itemsets of PAUUBI in *D*

1-itemsets	<i>E</i>
<i>auub</i>	120
<i>au</i>	120

7.2 Memory usage and number of candidate with different upper-bound threshold

The number of candidates of the compared algorithms is then evaluated, and the results are then shown in Fig. 4. Since the designed PRE-HAUI-DEL and the Apriori use the different framework to maintain and update the discovered information, it is meaningless to compare them in terms of number of the candidates. Thus, we only compare the Apriori algorithm with and without *lpub* strategy in this section. Note that an itemset is considered as a candidate if it is required to check the utility of it.

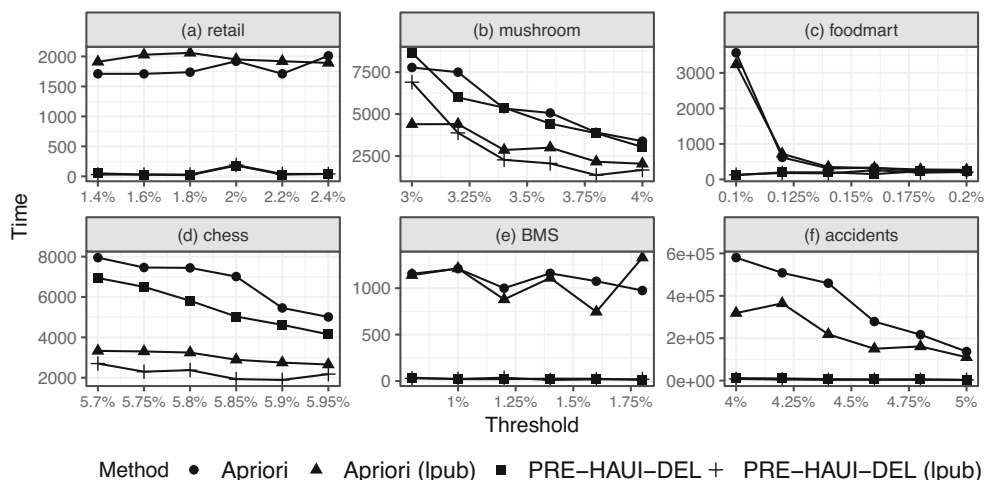
In the Section 7.3, it can be seen that lead partial upper bound (*lpub*) is more suitable for dense database, and from Fig. 4, it can be seen that there are fewer itemsets needed to be examined by using *lpub* in the dense databases, such as accidents, mushroom, and chess. This is because the *lpub* is a rigorous upper-bound and can reduce the size of candidates effectively. In a sparse database, it is not sensitive to be affected by shift values of upper-bounds. There might be a big utility cap between each possible itemset. Therefore, the influence between *pub* and *lpub* is not very obvious. To solve this issue, a dynamic self-adjusting technique should be developed in future works that can be used to select a suitable upper-bound for the current database during the mining process.

In addition, it can be seen from Fig. 5 that the Apriori uses more memory than Apriori (*lpub*). It shows that our new lead partial upper bound (*lpub*) strategy is effective for pruning the candidates in the search space. Moreover, the *Lpub* can cut more candidate sets and use less memory to store them. Secondly, we find that memory usage increases with the increase of candidates, because when there is more candidates generated, the more itemsets are stored in the memory.

Table 8 Characteristics of used datasets

Dataset	mushroom	foodmart	BMS	accidents	chess	retail
# <i>D</i>	8,124	21,557	59,602	340,183	3,196	88,162
# <i>I</i>	119	1,559	497	468	75	16,470
AvgLen	23	4	2.5	34	37	10
Type	dense	sparse	sparse	dense	dense	sparse

Fig. 3 The runtimes with different upper-bound utility thresholds



7.3 Scalability of database size

In this section, the runtimes of the proposed PRE-HAUI-DEL with different database sizes for deletion are then compared and discussed. Six different databases are utilized to verify the effectiveness of the proposed method. The experimental results are provided in Fig. 6.

In Fig. 6, it can be seen that the utilized *lpub* has better performance regarding the runtime cost. In most real datasets, the running time of Apriori (*lpub*) is less than that of the generic Apriori, especially in dense datasets, i.e., the mushroom and the chess databases. However, for instance, in both the foodmart and the retail datasets, the execution efficiency of Apriori (*lpub*) is not as good as that of Apriori. Thus, we can say that the *lpub* is more suitable for dense databases, but *lpub* does not play a better role in sparse databases. The main reason can be concluded that even the number of the candidates is reduced but the cost for performing the *lpub* strategy is high in some cases with

the additional calculation. Thus, the *lpub* is not suitable for sparse databases.

In addition, the designed PRE-HAUI-DEL could handle the decreasing process and reduce the runtime effectively. However, it can also be seen that the runtime of the PRE-HAUI-DEL is sometimes larger than Apriori. That is because some non-large and non-small itemsets are generated and maintained by using pre-large concept during the maintenance process. Moreover, PRE-HAUI-DEL algorithm needs to examine the updated utility value to achieve the final results for further maintenance or knowledge generation. Similarly, in these cases, the runtime of Apriori (*lpub*) is smaller than that of PRE-HAUI-DEL (*lpub*). In some cases, the performance of PRE-HAUI-DEL (*lpub*) is still better than that of PRE-HAUI-DEL, although its execution performance is worse than Apriori.

In Fig. 6a, we can see that the database size is reduced by fewer transactions each time, and in Fig. 6b, the database size is reduced by more transactions. As shown in Fig. 6a,

Fig. 4 The numbers of the candidates with different upper-bound utility thresholds

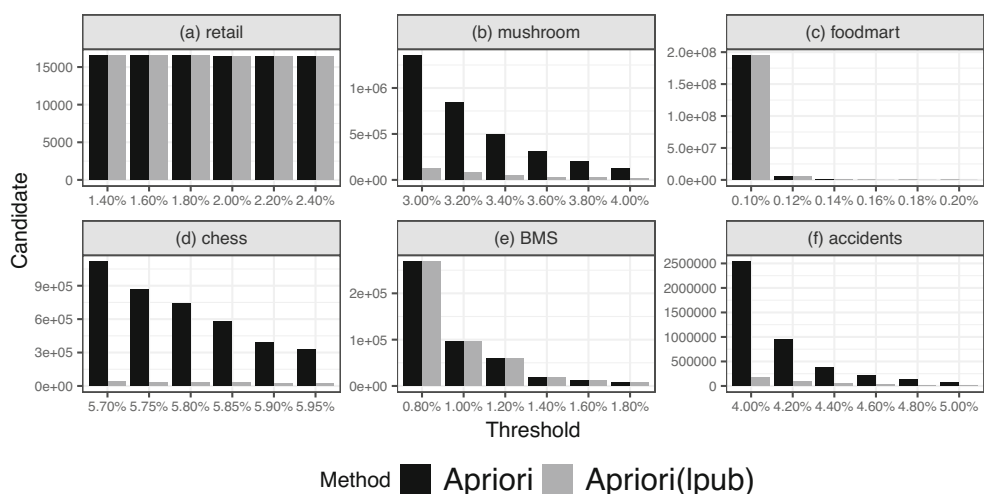
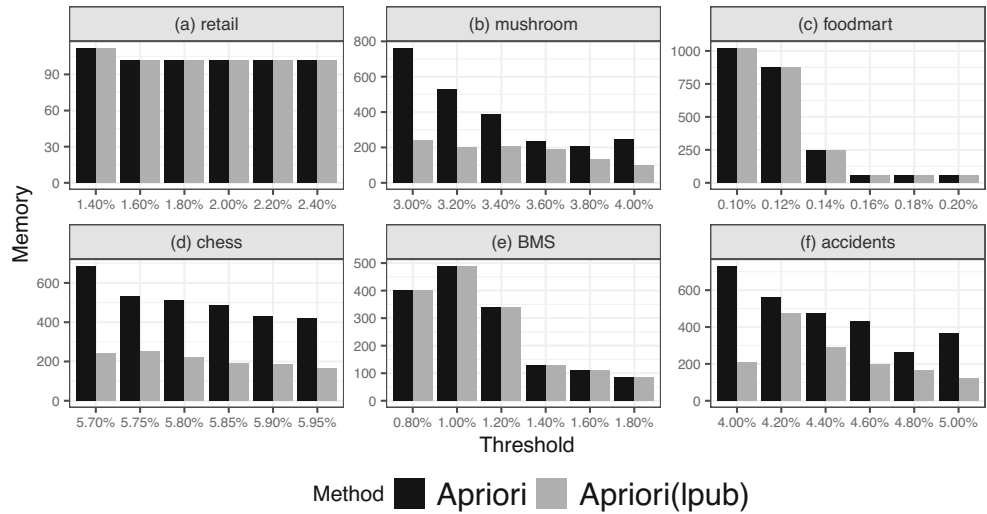


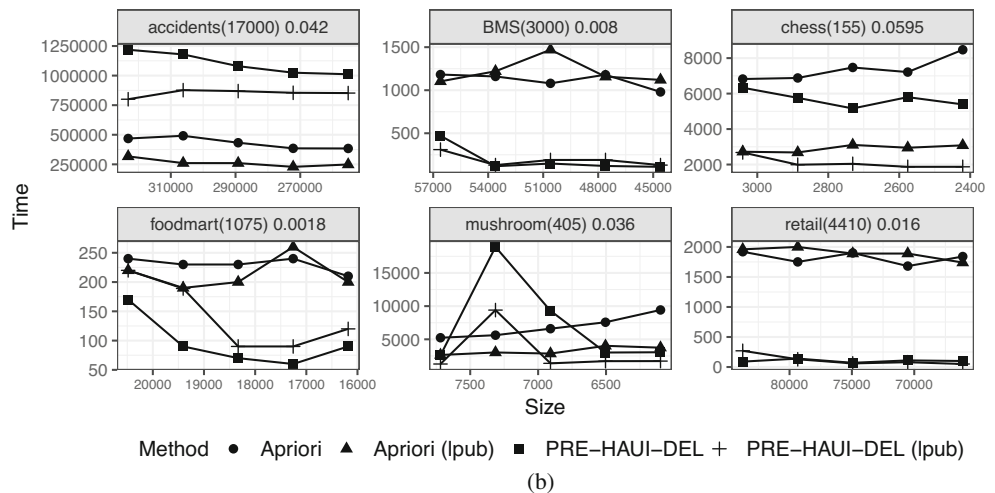
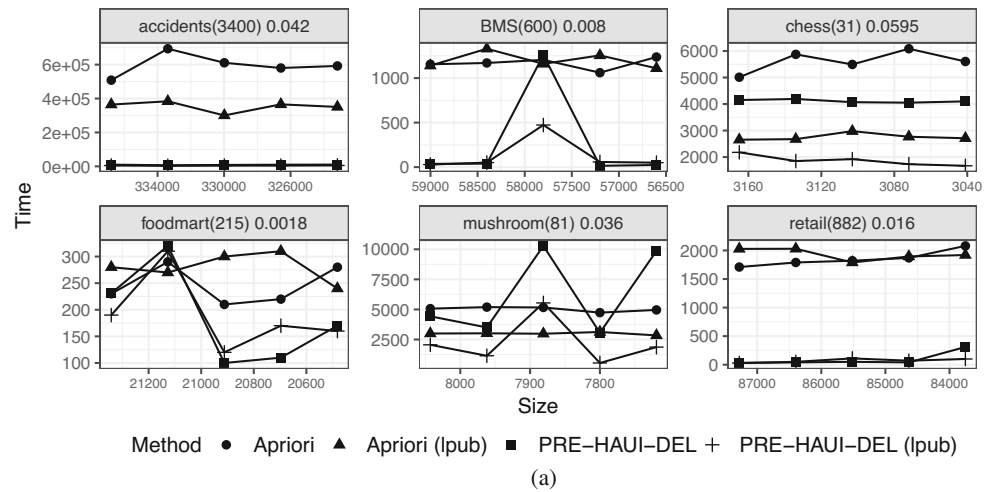
Fig. 5 The memory usage with different upper-bound utility thresholds



the re-scan operation will be performed after three times of transaction deletion, but the re-scan operation will be performed frequently in Fig. 6b. Based on (1), if the size of the decreased data is large, the safety bound will be

small; according to (1), the re-scanning process will be frequently performed; it causes that the pre-large concept is not suitable to be applied in a situation with a huge size of the deletion database. Another example can be observed

Fig. 6 The runtimes comparisons regarding different sizes for transaction deletion



from Fig. 6 that two designed PRE-HAUI-DEL algorithms showed good performance in the case of accidents database with the 3,400 transactions for deletion. When the size of deleted transactions grows up to 17,000 for each operation, the two PRE-HAUI-DEL algorithms will perform the re-scan process every time; this is because the *lpub* strategy of the over-estimated upper-bound value requires the extra computational cost, thus the developed PRE-HAUI-DEL requires more time than that of the Apriori approaches. Although this issue can be solved by increasing the size of the pre-large concept, but the developed model will suffer the huge computational cost in the updating progress. Thus, the proposed PE-HAUI-DEL with the pre-large concept is not suitable to be applied in the case by removing a very huge size of the transactions for deletion.

8 Conclusion and future work

In this paper, we first utilized pre-large concept into HAUIM for handling transaction deletion in dynamic databases. Besides, the *lpub* upper-bound model is also utilized in the developed algorithm that can be used to greatly reduce the size of the examined candidates in the search space. Compared to the generic model for updating the discovered knowledge in a batch mode, the designed PRE-HAUI-DEL can well maintain the discovered HAUIs without multiple database scans; the computational cost can be greatly reduced and the discovered knowledge regarding HAUIs can be maintained correctly and completely. In the future, we will then discuss the big data issue for HAUIM and also consider to apply different platforms (i.e., Hadoop or Spark) to handle the large-scale problems in HAUIM. At the same time, we will continuously study the concept of pre-large and apply it to the other mining tasks in dynamic databases, i.e., transaction modification. More effective model rather than the pre-large concept will be further discussed and studied in the future works.

Acknowledgments This research is supported by Shandong Provincial Natural Science Foundation (ZR201911150391).

Funding Open access funding provided by Western Norway University Of Applied Sciences.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted

use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Agarwal R, Srikant R (1994) Fast algorithms for mining association rules. In: International conference on very large data bases, vol 1215, pp 487–499
2. Agrawal R, Imieliński T, Swami A (1993) Mining association rules between sets of items in large databases. In: ACM SIGMOD International Conference on Management of Data, pp 207–216
3. Cheung DW, Han J, Ng VT, Wong C (1996) Maintenance of discovered association rules in large databases: An incremental updating technique. In: Proceedings of the Twelfth International Conference on Data Engineering, pp 106–114
4. Cheung DW, Lee SD, Kao B (1997) A general incremental technique for maintaining discovered association rules. In: Database systems for advanced applications', vol 97, pp 185–194
5. Deng ZH, Lv SL (2014) Fast mining frequent itemsets using nodesets. *Expert Syst Appl* 41(10):4505–4512
6. Erwin A, Gopalan RP, Achuthan N (2008) Efficient mining of high utility itemsets from large datasets. In: Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining, pp 554–561
7. Gan W, Lin JCW, Fournier-Viger P, Chao HC, Tseng VS, Yu PS (2019a) A survey of utility-oriented pattern mining. *IEEE Transactions on Knowledge and Data Engineering*
8. Gan W, Lin JCW, Fournier-Viger P, Chao HC, Yu PS (2019b) A survey of parallel sequential pattern mining. *ACM Trans Knowl Discov Data* 3(3):1–34
9. Han J, Pei J, Yin Y, Mao R (2004) Mining frequent patterns without candidate generation: a frequent-pattern tree approach. *Data Min Knowl Disc* 8(1):53–87
10. Hong TP, Wang CY, Tao YH (2001) A new incremental data mining algorithm using pre-large. *Intell Data Anal* 5(2):111–129
11. Hong TP, Lee CH, Wang SL (2011) Effective utility mining with the measure of average utility. *Expert Syst Appl* 38(7):8259–8265
12. Kim H, Yun U, Baek Y, Kim J, Vo B, Yoon E, Fujita H (2021) Efficient list based mining of high average utility patterns with maximum average pruning strategies. *Inf Sci* 543:85–105
13. Kim J, Yun U, Yoon E, Lin JCW, Fournier-Viger P (2020) One scan based high average-utility pattern mining in static and dynamic databases. *Futur Gener Comput Syst* 111:143–158
14. Lan GC, Hong TP, Tseng VS (2012) Efficient mining high average-utility itemsets with an improved upper-bound strategy. *Int J Inf Technol Decis Making* 11(5):1009–1030
15. Lin CW, Hong TP, Lu WH (2010) Efficiently mining high average utility itemsets with a tree structure. In: Asian Conference on Intelligent Information and Database Systems, pp 131–139
16. Lin CW, Hong TP, Lu WH (2011) An effective tree structure for mining high utility itemsets. *Expert Syst Appl* 38(6):7419–7424
17. Lin JCW, Ren S, Fournier-Viger P, Hong TP (2017a) Ehaupm: efficient high average-utility pattern mining with tighter upper-bounds. *IEEE Access* 5:12927–12940
18. Lin JCW, Ren S, Fournier-Viger P, Hong TP, Su JH, Vo B (2017a) A fast algorithm for mining high average-utility itemsets. *Appl Intell* 47(2):331–346
19. Ling Z, Zengrui T, Metawa N (2019) Data mining-based competency model of innovation and entrepreneurship. *J Intell Fuzzy Syst* 37(1):35–43
20. Liu J, Wang K, Fung BC (2012) Direct discovery of high utility itemsets without candidate generation. In: International Conference on Data Mining, pp 984–989

21. Liu J, Wang K, Fung BC (2015) Mining high utility patterns in one phase without generating candidates. *IEEE Trans Knowl Data Eng* 28(5):1245–1257
22. Liu M, Qu J (2012) Mining high utility itemsets without candidate generation. In: *International Conference on Information and Knowledge Management*, pp 55–64
23. Liu Y, Wk Liao, Choudhary A (2005) A two-phase algorithm for fast discovery of high utility itemsets. *Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, pp 689–695
24. Truong T, Duong H, Le B, Fournier-Viger P, Yun U (2019) Efficient high average-utility itemset mining using novel vertical weak upper-bounds. *Knowl-Based Syst* 104847:183
25. Tseng VS, Shie BE, Wu CW, Philip SY (2012) Efficient algorithms for mining high utility itemsets from transactional databases. *IEEE Trans Knowl Data Eng* 25(8):1772–1786
26. Wu JMT, Lin JCW, Tamrakar A (2019) High-utility itemset mining with effective pruning strategies. *ACM Trans Knowl Discov Data* 13(6):1–22
27. Wu JMT, Teng Q, Lin JCW, Yun U, Chen HC (2020) Updating high average-utility itemsets with pre-large concept. *J Intell Fuzzy Syst* 38(5):5831–5840
28. Yao H, Hamilton HJ, Butz CJ (2004) A foundational approach to mining itemset utilities from databases. In: *International Conference on Data Mining*, pp 215–221
29. Yen SJ, Lee YS (2007) Mining high utility quantitative association rules. *International Conference on Data Ware Housing and Knowledge Discovery*, pp 283–292
30. Yun U, Kim D, Yoo E, Fujita H (2018) Damped window based high average utility pattern mining over data streams. *Knowl-Based Syst* 144:188–205
31. Yun U, Nam H, Kim J, Kim H, Baek Y, Lee J, Yoon E, Truong T, Vo B, Pedrycz W (2020) Efficient transaction deleting approach of pre-large based high utility pattern mining in dynamic databases. *Futur Gener Comput Syst* 103:58–78
32. Zhao Z, Li C, Zhang X, Chiclana F, Viedma EH (2019) An incremental method to detect communities in dynamic evolving social networks. *Knowl-Based Syst* 163:404–415
33. Zida S, Fournier-Viger P, Lin JCW, Wu CW, Tseng VS (2017) Efim: a fast and memory efficient algorithm for high-utility itemset mining. *Knowl Inf Syst* 51(2):595–625

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Jimmy Ming-Tai Wu is currently an Assistant Professor at College of Computer Science and Engineering, Shandong University of Science and Technology. He was also an Assistant Professor at School of Computer Science and Technology, Harbin Institute of Technology-Shenzhen, China. His current research work is related to Data Mining, Big Data, Cloud Computing, Artificial Intelligence, Evolutionary Computation, Machine Learning

and Deep Learning. He was graduated and earned his Ph.D. degree with major in Computer Science and Engineering from National Sun Yat-sen University, Kaohsiung, Taiwan. He worked in an IC design company in Taiwan as a firmware developer and information technology manager in two years. He also was a research scholar in the Department of Computer Science and Information Engineering in National University of Kaohsiung, Kaohsiung, Taiwan and in the Department of Computer Science, College of Engineering in University of Nevada, Las Vegas.



Qian Teng is now a post-graduate in School of Computer Science and Engineering at Shandong University of Science and Technology, Qian-dao, China. Her research area is big data, and at present the main research is utility pattern mining.



Shahab Tayeb is a faculty with the Department of Electrical and Computer Engineering in the Lyles College of Engineering at California State University Fresno. Dr. Tayeb's research expertise and interests include network security and privacy, particularly in the context of the Internet of Things (IoT). His research incorporates machine learning techniques & data analytics approaches to tackle the detection of zero-day attacks. He has authored 45+

refereed research papers over the past four years, and his research findings have been highlighted by local, regional, and national agencies, including the National Science Foundation. He has also been the recipient of several scholarships and national awards, including a US Congressional Commendation for STEM mentorship.



Jerry Chun-Wei Lin received his Ph.D. from the Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan, Taiwan in 2010. He is currently a full Professor with the Department of Computer Science, Electrical Engineering and Mathematical Sciences, Western Norway University of Applied Sciences, Bergen, Norway. He has published more than 400 research articles in refereed journals (IEEE TKDE, IEEE TCYB,

IEEE TII, IEEE TITS, IEEE TNSE, IEEE TETCI, IEEE SysJ, IEEE SensJ, IEEE IOTJ, ACM TKDD, ACM TDS, ACM TMIS, ACM TOIT, ACM TIST) and international conferences (IEEE ICDE, IEEE ICDM, PKDD, PAKDD), 12 edited books, as well as 33 patents (held and filed, 3 US patents). His research interests include data mining, soft computing, artificial intelligence/machine learning, and privacy preserving and security technologies. He is the Editor-in-Chief of the International Journal of Data Science and Pattern Recognition, the Guest Editor/Associate Editor for several IEEE/ACM journals such as IEEE TFS, IEEE TII, ACM TMIS, ACM TOIT, and IEEE Access. He has recognized as the most cited Chinese Researcher respectively in 2018, 2019, and 2020 by Scopus/Elsevier. He is the Fellow of IET (FIET), senior member for both IEEE and ACM.

Affiliations

Jimmy Ming-Tai Wu¹ · Qian Teng¹ · Shahab Tayeb² · Jerry Chun-Wei Lin³ 

Jimmy Ming-Tai Wu
wmt@wmt35.idv.tw

Qian Teng
qrpe@foxmail.com

Shahab Tayeb
shahabtayeb@gmail.com

- ¹ School of Computer Science and Engineering, Shandong University of Science and Technology, Qingdao, China
- ² Department of Electrical and Computer Engineering, California State University, Fresno, USA
- ³ Department of Computer Science, Electrical Engineering and Mathematical Sciences, Western Norway University of Applied Sciences, Bergen, Norway