Applied Intelligence (2020) 50:3788–3807
https://doi.org/10.1007/s10489-020-01743-y

Incrementally updating the high average-utility patterns with pre-large concept

Jerry Chun-Wei Lin[1] · Matin Pirouz[2] · Youcef Djenouri[3] · Chien-Fu Cheng[4] · Usman Ahmed[1]

Published online: 30 June 2020

Abstract
High-utility itemset mining (HUIM) is considered as an emerging approach to detect the high-utility patterns from databases. Most existing algorithms of HUIM only consider the itemset utility regardless of the length. This limitation raises the utility as a result of a growing itemset size. High average-utility itemset mining (HAUIM) considers the size of the itemset, thus providing a more balanced scale to measure the average-utility for decision-making. Several algorithms were presented to efficiently mine the set of high average-utility itemsets (HAUIs) but most of them focus on handling static databases. In the past, a fast-updated (FUP)-based algorithm was developed to efficiently handle the incremental problem but it still has to re-scan the database when the itemset in the original database is small but there is a high average-utility upper-bound itemset (HAUUBI) in the newly inserted transactions. In this paper, an efficient framework called PRE-HAUIMI for transaction insertion in dynamic databases is developed, which relies on the average-utility-list (AUL) structures. Moreover, we apply the pre-large concept on HAUIM. A pre-large concept is used to speed up the mining performance, which can ensure that if the total utility in the newly inserted transaction is within the safety bound, the small itemsets in the original database could not be the large ones after the database is updated. This, in turn, reduces the recurring database scans and obtains the correct HAUIs. Experiments demonstrate that the PRE-HAUIMI outperforms the state-of-the-art batch mode HAUI-Miner, and the state-of-the-art incremental IHAUPM and FUP-based algorithms in terms of runtime, memory, number of assessed patterns and scalability.

Keywords High average-utility itemset · Insertion · Pre-large · Dynamic · AUL-structures

1 Introduction

Association-rule mining (ARM) [1, 6–8, 15] is the most popular method to discover the relationship among the itemsets from databases, where the potential and implicit information can be discovered and revealed. Several algorithms of ARM were developed and most of them rely on the generate-and-test approach, which is the same as the traditional Apriori algorithm [1]. It uses two thresholds to verify whether an itemset is a frequent one in the first stage under the support threshold and generate the set of association rules in the second stage under the confidence threshold. This approach needs higher computational cost to generate the candidate set in a level-wise manner. Also, this process has a significant memory cost by keeping the promising candidates. To mitigate this limitation, the FP-growth [15] was presented to build the Frequent-Pattern (FP)-tree structure and developed an FP-growth mining approach to discover the frequent itemsets without candidate generation. Several variants were also respectively studied and discussed to efficiently mine the desired information from static databases.

ARM only treats the database as a binary one, therefore ignoring the other factors such as interestingness or weights. This situation may cause information loss since the database may come from a heterogeneous environment and each transaction may consist of several factors. High utility itemset (pattern) mining (HUIM or HUPM) [19, 38] is a variant of the ARM, which focuses on considering unit profit and quantity of the items to mine the set of high utility itemsets (HUIs). The user first defines the threshold and if the utility of an itemset is greater than the threshold, an itemset is treated as a HUI. This

✉ Jerry Chun-Wei Lin
jerrylin@ieee.org

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

approach ignores the frequency factor but brings more profitable itemsets to the retailer or decision-maker, which is more suitable in real-life applications. Since the traditional HUIM cannot obtain the downward closure property, many candidates should be generated for obtaining the actual HUIs. To overcome this drawback, a two-phase approach called TWU model [20] was presented, which utilizes the high transaction-weighted utilization itemsets (HTWUIs) by maintaining the transaction-weighted downward closure (TWDC) property. Since the upper-bound value of a HTWUI was estimated, the size of the candidates can be greatly reduced as compared to the traditional "combinational generation". Lin et al. then developed the HUP-tree structure [25], which keeps the 1-HTWUIs in a compressed tree, thus speeding up the mining performance. Liu then proposed a novel utility-list (UL)-structure [27] to efficiently mine the HUIs from the databases based on the TWU model. The UL-structure uses the join operation to produce the *k*-itemsets for HUIM. The experimental results illustrate that the UL-structure has better performance than that of the Aprori-like or pattern-growth models. Several extensions of the TWU model were presented and some of them are still work in progress [26, 30, 35, 39].

Although the HUIM uncovers a more realistic problem and highlights worthy itemsets for decision-making, it still suffers from a major problem. A larger itemset will have a higher utility. Thus, an itemset is considered as a HUI if its subset contains an item with a very high utility value; any combinations with this high-utility item (i.e., diamond or caviar) will be treated as the HUI. To solve this limitation, a more balanced scale called high average-utility itemset mining (HAUIM) [17] was developed, which detects the average-utility of an itemset irregardless of its size. An average-utility of an itemset calculates the ratio of the utility over the number of items. This method creates an alternative choice for decision-making with the consideration of the itemset size. Hong et al. designed the TPAU algorithm [17] to identify the set of high average-utility itemsets (HAUIs). It uses the average-utility-upper-bound (*auub*) model to estimate the upper-bound value on the high average-utility-upper-bound itemset (HAUUBI), thus maintaining the downward closure property which reduce the size of the candidates. Lin et al. then designed a HAUP-tree framework [24] that keeps the 1-HAUUBIs in a condense tree form. This approach is capable of mining the set of HAUIs without creating new candidates. It is based on the pattern-growth approach but each node in the tree structure keeps an additional array for further information, i.e., the quantity values of its prefix items in the path. The computational cost of the multiple database scans can thus be reduced. An efficient average-utility (AU)-list framework [31] was also developed to speed up mining performance of the HAUIs. Based on the simple join operation ,the required information of HAUIs can be easily retrieved and discovered without candidate generation. Many extensions were also discussed to efficiently mine the set of HAUIs [32, 40, 41].

The aforementioned approaches consider the problem of mining from a static database. Traditional algorithms handle dynamic databases in a batch manner. For example, as the database grows and transactions are concatenated to the original database, the updated database should be processed. The database will go through re-scanning and the up-to-date knowledge should be re-mined. Cheung et al. developed a fast updated (FUP) concept [5] which stores the frequent itemsets in a dynamic database. With a change in the database, their framework considers four scenarios, and each scenario treats the update differently following a prescribed approach. In the past, the FUP-based concept was utilized in the ARM [5, 16], HUIM [22, 23], and HAUIM [33, 36], but those approaches still face a limitation such that some itemsets are still required to be re-scanned and an additional database scan is necessary for maintaining those itemsets. Wu et al. [37] presented a level-wise approach by adopting the pre-large concept for incremental mining in HAUIM. However, this model lacks of the theroical proofs to show that the pre-large can successfully hold the correctness and completeness of the maintained HAUIs. Moreover, this level-wise approach still requires a huge number of computational cost. This research applies the pre-large concept [14] for maintaining the itemsets in the dynamic database with transaction insertion for HAUIM. An algorithm called PRE-HAUIMI is designed for transaction insertion. The major contributions of this paper include the following.

- We first propose the theoretical definitions, theorems and proofs to show that the pre-large concept (pre-large average-utility upper-bound itemsets, PAUUBI) can be greatly utilized in the high average-utility itemset mining (HAUIM) by holding the completeness and correctness to maintain the discovered high average-utility itemsets.
- The average-utility-list (AUL)-structure is applied to obtain the 1-HAUUBIs, and a new AUL-structure is also used to keep the 1-PAUUBIs for the maintenance purpose. Thus, the simple join operation can be easily applied to generate the *k*-itemsets of the HAUIs, outperforming the level-wise approach. Moreover, an enumeration tree is built to determine whether the supersets of a tree node should be explored, thus reducing the join operation performing on the AUL-structures of the unpromising candidates.
- Extensive experimentation is performed to measure the effectiveness of the proposed PRE-HAUIMI as compared to the state-of-the-art HAUI-Miner in the

batch mode, and two state-of-the-art IHAUPM and FUP-based algorithms running in the incremental maintenance. The performance is measured in terms of runtime, memory, number of assessed patterns, and scalability.

2 Literature review

In this section, existing works related to high average-utility itemset mining and incremental mining are studied and reviewed as below, respectively.

2.1 High average-utility itemset mining

Traditional association-rule mining (ARM) considers the occurrence frequency of the itemsets only to mine the relationship of the itemsets in the binary databases. Apriori is the fundamental algorithm that applies generate-and-test approach to level-wise mine the association rules (ARs) based on the minimum support and minimum confidence thresholds. Furthermore, the Apriori algorithm maintains the downward closure (DC) property to keep the correctness and completeness of discovered ARs. This DC property is also applied to many other tasks of knowledge discovery such as sequential pattern mining (SPM) [11] or weighted pattern mining (WPM) [13]. To improve the mining performance, the frequent-pattern (FP)-tree structure was presented to keep only the frequent 1-itemsets in the tree structure. A recursive FP-growth mining algorithm was then implemented to mine the frequent *k*-itemsets from the FP-tree. Since the DC property is utilized in the FP-tree structure by keeping the frequent 1-itemsets, thus the completeness and correctness can be held and maintained. However, the relevant works of ARM or frequent itemset mining only consider the frequency of the itemsets, thus may mislead the decision-making due to the insufficient information.

High utility itemset mining (HUIM) [19, 38] is gaining popularity as it takes into account the unit profit as well as the quantity of the items as the major factors to discover the set of HUIs. An itemset is considered to be a HUI if it meets the condition of having a higher utility than a set threshold. Traditional algorithms of HUIM face the “combinational explosion” problem since the naive HUIM does not hold the downward closure property. Thus, the transaction-weighted utility (TWU) model was proposed to solve this limitation by taking an upper-bound value to evaluate the set of high transaction-weighted utilization itemsets (HTWUIs) against the required high-utility quantity, thus the downward closure property to ensure the completeness and the correctness of the discovered HUIs can be maintained. Lin et al. then presented the high-utility-pattern (HUP)-tree [25] to speed up the mining performance based on the FP-tree structure. An extra array of each tree node is then built to keep the quantity values of the prefix path, thus the mining performance can be greatly improved. However, this progress needs more memory usage to keep the extra information, which is not sufficient to handle a big dataset. Tseng et al. then presented the UP-growth+ [35] to build the similar structure as FP-tree but applied several pruning strategies to early reduce the unpromising candidates in the search space. This algorithm showed better performance than the standard TWU model. To better improve the mining performance, Liu et al. [27] then developed the utility-list (UL)-structure to keep the necessary information based on the TWU model. The UL-structure adopts a simple join operation to produce the *k*-itemsets, and since the 1-HTWUIs are kept in the UL-structure, the correctness and completeness can thus be maintained by the HUI-Miner. Several extensions [3, 4, 26, 30] based on the TWU model were respectively developed and the state-of-the-art approach for HUIM is called EFIM [42] that relies on two new upper bounds to reduce the search space of the unpromising candidates,. Moreover, a new array-based utility counting technique was investigated to calculate the upper bounds, and the projection and merging techniques were also developed to improve mining performance. Besides the above mining algorithms that were developed for standard HUIM, the varied knowledge based on the utility-oriented pattern mining were also developed and discussed. For example, the top-k HUIM [18] is used to mine the top-*k* HUIs instead of mining the complete HUIs from the database; closed HUIM [10, 34] is used to mine the closed HUIs from the database; and CoUPM [12] is used to consider the correlation of the HUIM. Those algorithms were developed to provide fewer utility-oriented patterns for decision-making, which is more effective for on-line decision-making.

High average-utility itemset mining (HAUIM) [17] extends HUIM on the itemset length for evaluation. This is because most algorithms of HUIM under-perform when both the size of an itemset and the utility of an itemset increase. Thus, for example, a bread may be considered as a HUI if it always appears with caviar in basket analytics. The TPAU algorithm [17] was the first algorithm in HAUIM and it is based on the Apriori. It uses the (*auub*) framework to keep a downward closure property. Lin et al. developed the HAUP-tree [24] to maintain the satisfied 1-itemsets in the tree structure. An array to keep the quantities of its prefix items in the path is also attached to each node in the HAUP-tree, thus speed up the mining performance by reducing the multiple database scans. Although this method is more efficient than that of the traditional generate-and-test approach [17], it still needs to keep a huge memory usage for the mining progress. Thus, a HAUI-Miner [31] was presented to adopt the average-utility-list structure to mine the HAUIs without generating new candidates. This is the state-of-the-art model to efficiently mine the HAUIs from the static databases. Several extensions such as PBAU [28] and HAUI-Tree [29] were respectively studied and most of them rely on the *auub* model to find the HAUI set.

2.2 Incremental mining

Traditional association-rule mining (ARM) [1, 15] aims at mining the set of association rules from a binary database and only considers a static database, which indicates that the size of the database is unchanged. Many extensions of ARM are presented [6, 7, 21, 39], which emphasize on improving the mining performance in static databases. However, in a realistic environment, the database size is not stable; as new transactions are added. In this situation, the already discovered information or rules should be updated since some rules may be arisen or missed. Adding the new information is critical since it sometimes influences the decisions or strategies of a company/industry in a significant manner. Traditional algorithms in pattern mining handle this situation using a batch approach. This re-scans the new database with a addition or removal of data point. This progress is a computational cost, and the discovered information in the past becomes useless.

To better use the discovered information, Cheung et al. developed the Fast UPdate (FUP) concept [5] to deal with transaction updates for transaction insertion and thereby keeping the frequent itemsets in a dynamic database. The FUP concept divides the itemsets into four subsets with respect to the original database and new transactions. Each case has its own developed model to efficiently update and maintain the discovered frequent itemsets according to whether it exists or not in the original database or in the newly inserted transactions. This approach has been utilized to ARM [5, 16], HUIM [22, 23], and HAUIM [33, 36]. However, some itemsets are required to be assessed since they were not kept from the original database. For example, if an itemset exists in case 3 that indicates this itemset was not a frequent itemset in the original database but is considered as the frequent itemset in the newly inserted transaction, the original database is required to be rescanned to keep the integrity of the knowledge, especially in maintaining the correctness for the updated databases. Thus, this approach showed lower efficiency when many itemsets in case 3 are required for maintenance.

To address this shortcoming, Hong et al. applied the pre-large concept [14]. The pre-large concept utilized two thresholds for maintaining large and pre-large itemsets. This way, the original database and the new transactions are divided into nine cases, and several procedures were developed to handle the itemsets of each case. The pre-large itemsets play the role of a placeholder to avoid the direct transition of an itemset to/from large to small. An equation was also designed to ensure that as long as there is a change in size within a threshold, there is no need for re-computation (case 7) and the correctness and completeness of the discovered frequent itemsets can still be obtained. The summary of the pre-large concept with transaction insertion is shown in Fig. 1.

Cases 1, 5, 6, 8, and 9 would not change the results in a meaningful way as a result of the weighted average of their frequencies. Some rules may be removed if they appear in cases 2 and 3, and some rules may be arisen if they appear in cases 4 and 7. Since we keep large and pre-large itemsets from the original database, the itemsets in cases 2, 3 and 4 are easily handled. Based on the defined equation for the safety bound, case 7 itemsets are always smaller than the the threshold. The safety bound (f) is shown as:

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

where f indicates the safety bound, S_u is the upper bound, S_l is the lower bound, and $|D|$ is the size of the initial database (number of transactions).

3 Preliminaries and problem statement

Let D contain n transactions where $D = \{T_1, T_2, \ldots, T_n\}$, and d is the set of new transactions. Each transaction T_q in D or d contains several distinct items such that $T_q = \{i_1, i_2, \ldots, i_k\}$, and I is defined as the collection of all items appearing in D and d such that $I = \{i_1, i_2, \ldots, i_m\}$. Therefore, $i_j \in I$, and $T_q \subseteq D$ or $T_q \subseteq d$. A profit table is defined as $utable = \{p(i_1), p(i_2), \ldots, p(i_m)\}$, which contains the item profit values. A minimum HAU threshold is represented with δ, set by the user. Tables 1 and 2 illustrate an example of the initial dataset and profit table, respectively.

Table 1 A running example of the original database

TID	Items:quantities
T_1	a:1, b:5, c:2, d:3, f:6
T_2	b:2, c:3, e:2
T_3	a:1, b:2, d:1, f:1
T_4	a:1, c:3, d:2
T_5	a:1, e:1
T_6	b:7, d:1, f:2
T_7	a:3, b:9, c:3, d:1

Table 2 An unit profits of the items

Item	*a*	*b*	*c*	*d*	*e*	*f*
Profit	4	1	5	7	6	3

Definition 1 The average-utility of an item i_j in a transaction T_q can be represented by $au(i_j, T_q)$:

$$au(i_j, T_q) = \frac{q(i_j, T_q) \times p(i_j)}{1}, \tag{2}$$

where $q(i_j, T_q)$ is the quantity of i_j in T_q, and $p(i_j)$ is the unit profit value of i_j.

For example, the average-utility *au* of an item (*a*) in T_1 is $au(a, T_1) = \frac{1\times4}{1}$ (= 4).

Definition 2 The *au* of a *k*-itemset *X* in a transaction T_q is represented by $au(X, T_q)$:

$$au(X, T_q) = \frac{\sum_{i_j \subseteq X \wedge X \subseteq T_q} q(i_j, T_q) \times p(i_j)}{|X|} = \frac{\sum_{i_j \subseteq X \wedge X \subseteq T_q} q(i_j, T_q) \times p(i_j)}{k}, \tag{3}$$

where *k* is the number of items in *X*.

For example, the *au* of the itemset (*ab*) in T_1 is $au(ab, T_1) = \frac{1\times4+5\times1}{2}$ (= 4.5).

Definition 3 The *au* of an itemset *X* in *D* is represented by $au(X)$:

$$au(X) = \sum_{X \subseteq T_q \wedge T_q \in D} au(X, T_q). \tag{4}$$

For instance, the *au* of the itemset (*ab*) in *D* is calculated as $au(ab) = au(ab, T_1) + au(ab, T_3) + au(ab, T_7)$ (= 4.5 + 3 + 10.5)(= 18).

Definition 4 The transaction utility of a transaction T_q can be represented by $tu(T_q)$:

$$tu(T_q) = \sum_{i_j \subseteq T_q} u(i_j, T_q). \tag{5}$$

For instance, the transaction utility of T_1 is calculated as (1× 4 + 5× 1 + 2× 5 + 3× 7 + 6× 3)(= 58).

Definition 5 The total utility *TU* of a database *D* can be represented by TU^D:

$$TU^D = \sum_{T_q \in D} tu(T_q). \tag{6}$$

For example, *TU* of Table 1 is TU^D (= 58 + 29 + 16 + 33 + 20 + 43)(= 209).

Definition 6 An itemset is defined as a HAUI if its *au* meets:

$$HAUI \leftarrow \{X | au(X) \geq TU^D \times \delta\}. \tag{7}$$

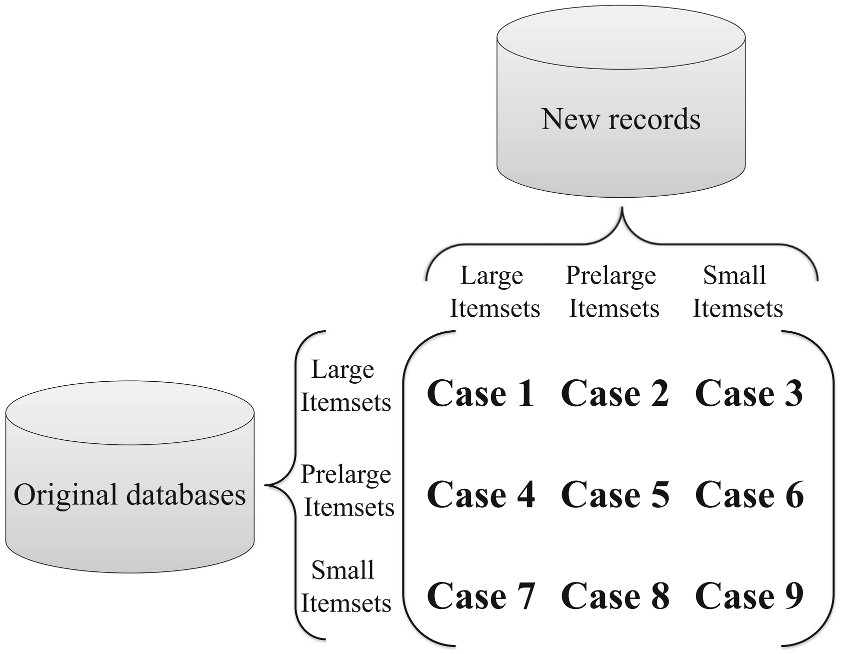


Fig. 1 Nine cases of the pre-large concept

As an instance, suppose the δ is 20%, then the minimum au is calculated as (209× 0.2) (= 41.8). In this example, (c) and (cd) are considered as the HAUI since $u(c)$(= 55), and $u(ac) = \frac{82}{2}$ (= 41), but (a) is not since its average-utility is calculated as $u(a)$ (= 28).

To calculate the downward closure property for removing the unlikely ones, the ($auub$) model [17] over-estimates the itemsets utility.

Definition 7 The transaction-maximum utility tmu of a transaction T_q is represented by $tmu(T_q)$:

$$tmu(T_q) = \max\{u(i_j) | i_j \subseteq X \wedge X \subseteq T_q\}. \tag{8}$$

As an instance, the *tmu* of T_1 is *max*{4, 5, 10, 21, 18} (= 21).

Definition 8 The average-utility upper-bound $auub$ of an itemset X from the initial database can be represented by $auub(X)^D$:

$$auub(X)^D = \sum_{X \subseteq T_q \wedge T_q \in D} tmu(T_q), \tag{9}$$

where $tmu(T_q)$ is the maximum utility of transaction T_q given $i_j \subseteq X \wedge X \subseteq T_q$.

For example, the *auub* value of an item (a) is calculated as $auub(a)$(= 21 + 7 + 15 + 6 + 15)(= 64).

Property 1 (Downward closure of *auub*) For an itemset Y as the superset of an itemset X under $X \subseteq Y$, we have:

$$auub(Y)^D \leq auub(X)^D. \tag{10}$$

Therefore, if $auub(X)^D \leq TU^D \times \delta$ then $auub(Y)^D \leq auub(X)^D \leq TU^D \times \delta$ is true for all supersets of X.

Definition 9 (High average-utility upper bound itemset, HAUUBI) An itemset X is a HAUUBI given that:

$$HAUUBI^D \leftarrow \{X | auub(X)^D \geq TU^D \times \delta\}. \tag{11}$$

As an example, the itemsets (a) and (ad) are considered as a HAUUBI since their *auub* values are respectively calculated as $auub(a)$(= 64 > 41.8), and $auub(ad)$(= 58 > 41.8), but the itemset (e) is not considered as HAUUBI since $auub(e)$(= 21 < 41.8).

For incremental mining, let us consider the case where there are several addition to the database (see Table 3). With this, the problem is defined as:

Problem statement In reality, the size of the historical data may exceed a million zeta-bytes, for example, a supermarket with an annual record of the purchase products from customers. However, a newly record for a month is going to be merged with the past records forming an updated database. In this situation, the generic and batch-mode approach is to directly integrate those two databases and re-mine the updated records again for retrieving the up-to-date information. This is not reasonable since the past computational cost and discovered information become useless. We are interested in efficiently holding and appending the initial database with the new additions, without re-scanning the new database. In this paper, we focus on maintaining the discovered HAUIs for transaction insertion, which is a real case in the retail industry. For the incremental HAUIM, it is necessary to design an efficient algorithm for transaction insertion to maintain and update the discovered information, and the multiple scans of the updated database should be avoided. Accordingly, an HAUI itemset is:

Table 3 A new inserted database

TID	Items:quantities
T_8	a:1, e:10, f:5
T_9	c:1, d:3
T_{10}	b:1

$$HAUI \leftarrow \{X | au(X)^U \geq (TU^D + TU^d) \times S_u\}, \tag{12}$$

where $au(X)^U$ indicates the updated average-utility of X, and TU^d as the total utility in the inserted database d. Moreover, S_u is a user-specified upper-utility threshold, and is adjustable by domain experts.

4 Proposed PRE-HAUIMI framework for transaction insertion

Traditionally, various frameworks were designed to speed up the process of finding HUIM, and the well-known structure is called utility-list (UL)-structure. The HAUI-Miner [31] was developed to find the HAUIs by utilizing the UL-structure, and presented an average-utility-list (AUL)-structure, running in the batch mode. Lin et al. developed the FUP-based approach [36] by using the AUL-structure, but it still needs, however, multiple database scans if some itemsets belong to the Case 3. In this paper, we adopt the AUL-structure and utilize the pre-large concept for handling the incremental problem of HAUIM. Based on the utilized pre-large concept, the designed PRE-HAUIMI handles the sets of 1-HAUUBIs pre-large average-utility upper-bound 1-itemsets (1-PAUUBIs) efficiently by adopting the AUL-structure, and the 1-PAUUBIs can successfully play as the buffer to reduce the condition such as from HAUUBI directly to small and vice versa. An upper-utility threshold is set as S_u and the lower-utility is set as S_l for the designed

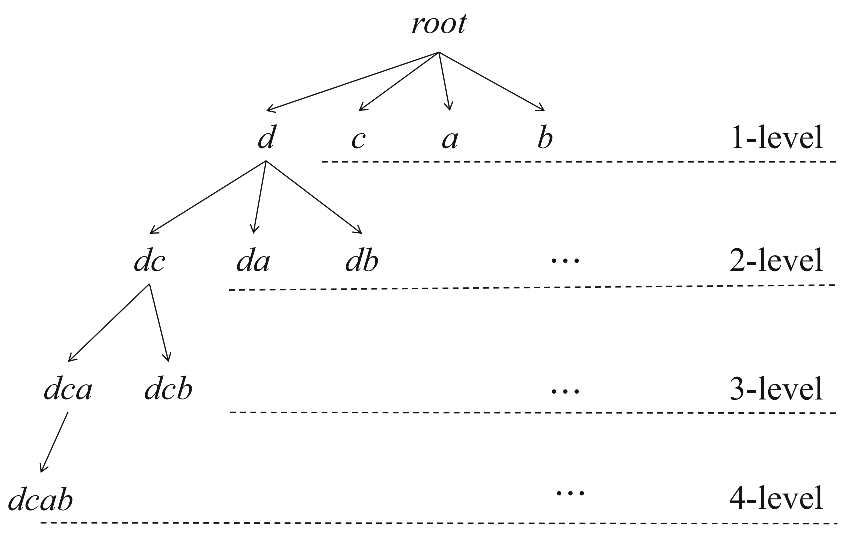


Fig. 2 A sample enumeration tree

PRE-HAUIMI. Notice that the S_u is equal to the high average-utility threshold used in traditional HAUIM. The details of PRE-HAUIMI are described below:

Definition 10 (Pre-large average-utility upper-bound itemset, PAUUBI) An itemset X is a pre-large average-utility upper-bound itemset (PAUUBI) in the initial database given:

$$PAUUBI^D \leftarrow \{X | S_l \times TU^D \leq auub(X)^D < S_u \times TU^D\}. \tag{13}$$

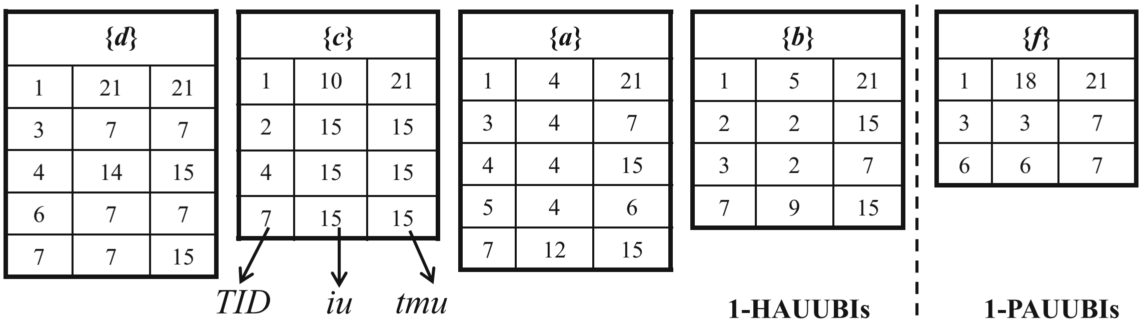
For instance, suppose the S_u and S_l are respectively set as 21% and 16% in the running example. The itemset (*ab*) is PAUUBI with an *au* of 34, which lies between S_l (= 209 × 16%)(= 33.4) and S_u (= 209 × 21%) (= 43.9).

4.1 The Average-Utility-List (AUL)-structure

The HAUI-Miner algorithm [31] uses AUL-structures to keeps three fields for later mining progress. The first field is the transaction ID (*tid*). The utility of an item i_j of each *tid* is shown as *iu*, and the transaction-maximum-utility of an item i_j of each *tid* is stated as *tmu*. The AUL-structure in the designed PRE-HAUIMI keeps not only the 1-HAUUBIs but also the 1-PAUUBIs. Although this process requires the extra memory usage, it reduces the need for re-scans and the completeness and correctness can be exactly obtained. For the construction progress of the AUL-structure, if *auub* is not more than the upper-utility threshold, it can be denoted as 1-HAUUBI, and if the *auub* is less than the upper-utility threshold but greater than the lower-utility threshold, it is then considered as the 1-PAUBBI.

To create the k-itemset ($k \geq 2$) of the potential HAUIs, an enumeration tree is then explored by the depth-first search (DFS). DFS decides between proceeding with the superset (k+1)-itemsets of k-itemsets or skipping it. For supersets that meet the requirement, an iterative join process creates the AUL-structures of k-itemsets. This is done for all remaining candidates. Finally, the final HAUIs can be revealed by scanning the database. From the given example, the enumeration tree is shown in Fig. 2.

Property 2 All the satisfied 1-HAUUBIs in the initial database are put into an ascending order based on their *auub*. This facilitates the transition to the AUL structures.



{*d*}		
1	21	21
3	7	7
4	14	15
6	7	7
7	7	15

{*c*}		
1	10	21
2	15	15
4	15	15
7	15	15

{*a*}		
1	4	21
3	4	7
4	4	15
5	4	6
7	12	15

{*b*}		
1	5	21
2	2	15
3	2	7
7	9	15

{*f*}		
1	18	21
3	3	7
6	6	7

TID
iu
tmu
1-HAUUBIs
1-PAUUBIs

Fig. 3 The constructed AUL-structures of the original database

Property 3 In order to efficiently hold the correctness and the completeness of the discovered HAUUBIs and HAUIs, every item should go through the AUL-structures. The negligible size of this process as compared to the initial database in real-world scenarios creates an advantage.

Property 4 The ordered 1-HAUUBIs and 1-PAUUBIs in the new database is closely related to the sorted order from the initial database. This is rooted in the smaller size of the new one as compared to the initial dataset. Consequently, the order stays in tact for the most part throughout the process.

Due to the preservation of 1-HAUUBIs and 1-PAUUBIs in the formation of the AUL-structures, the proposed PRE-HAUIMI is capable of mining HAUUBIs and HAUIs while bypassing the candidate generation. The correctness and completeness of the updated sets remain complete. The results of the AUL-structures of 1-HAUUBIs and 1-PAUUBIs are illustrated in Fig. 3.

4.2 The utilized pre-large concept

The developed PRE-HAUIMI divides the itemsets into nine cases based on the utilized pre-large, as adapted from [14]. Through further procedure, 1-HAUUBIs and 1-PAUBBIs are identified and kept in store. Figure 4 gives an overview of the nine cases of the utilized pre-large concept.

- **Case 1**: Given a HAUUBI that is present both initial database and updated transaction, the HAUUBI stays as is.
- **Case 2**: Given a HAUUBI that is present in the initial database but become a PAUUBI in updated transaction, the HAUUBI may stay in tact or transform as a PAUUBI.
- **Case 3**: Given a HAUUBI that is present in the initial database but a small itemset in the updated transaction, the HAUUBI may stay in tact or transform as a PAUUBI or a small itemset.
- **Case 4**: Given a PAUUBI that is present in the initial database but become a HAUUBI in updated transaction, the PAUUBI may stay in tact or transform as a HAUUBI.
- **Case 5**: Given a PAUUBI that is present both initial database and updated transaction, the PAUUBI stays as is.
- **Case 6**: Given a PAUUBI that is present in the initial database but a small itemset in the updated transaction, the PAUUBI may stay in tact or transform as a small itemset.
- **Case 7**: Given a small itemset that is present in the initial database but a HAUUBI in the updated transaction, the small itemset may stay in tact or transform as a PAUUBI. The latter occurs when the total utility in the inserted transactions is less than the threshold.
- **Case 8**: Given a small itemset that is present in the initial database but a PAUUBI in the updated transaction, the small itemset may stay in tact or transform as a PAUUBI.
- **Case 9**: Given a small itemset that is present both initial database and updated transaction, the small itemset stays as is.

Accordingly, the theoretical foundation of the safety bound is defined by:

$$f = \frac{(S_u - S_l) \times TU^D}{1 - S_u} \tag{14}$$

Thus, case 7 cannot become a HAUUBI after the update if (TU^d) stays under (f).

Theorem 1 *Consider S_l and S_u as the lower-utility and the upper-utility bounds, and TU^D and TU^d be the total utility in the initial database and the updated one. Thus, if $TU^d \leq f\ (= \frac{(S_u - S_l) \times TU^D}{1 - S_u})$, case 7 holds true.*

Proof From $TU^d \leq \frac{(S_u - S_l) \times TU^D}{1 - S_u}$, we can obtain that:

$TU^d \leq \frac{(S_u - S_l) \times TU^D}{1 - S_u}$
$\Rightarrow TU^d \times (1 - S_u) \leq (S_u - S_l) \times TU^D$
$\Rightarrow TU^d - TU^d \times S_u \leq TU^D \times S_u - TU^D \times S_l$
$\Rightarrow TU^d + TU^D \times S_l \leq S_u \times (TU^D + TU^d)$
$\Rightarrow \frac{TU^d + TU^D \times S_l}{TU^D + TU^d} \leq S_u$

For an itemset X in case 7, if it is a small itemset in the initial database, we have:

$$auub^D(X) < S_l \times TU^D. \tag{15}$$

Also, it holds the following equation in the new transactions as:

$$TU^d > auub(X)^d > S_u \times TU^d > S_l \times TU^d. \tag{16}$$

Thus, for an itemset X, its utility ratio in the new database U is $\frac{auub(X)^U}{TU^D + TU^d}$, also expanded as:

$$\frac{auub(X)^U}{TU^D + TU^d} = \frac{auub(X)^D + auub(X)^d}{TU^D + TU^d} < \frac{S_l \times TU^D + TU^d}{TU^D + TU^d}. \tag{17}$$

From the above equations, we can conclude that:

$$\frac{auub(X)^U}{TU^D + TU^d} < \frac{S_l \times TU^D + TU^d}{TU^D + TU^d} \leq S_u. \tag{18}$$

Thus, the *auub* of an itemset X is small in the new database as long as the updated total utility is not larger than $\frac{(S_u - S_l) \times TU^D}{1 - S_u}$. □

Based on the above theorem and proof, we can ensure that the multiple database scans can be greatly avoided if the total utility in d is no larger than the safety bound (f). Thus, Case 7 holds true.

For this example, let us consider the upper-utility threshold as S_u (= 21%), and lower-utility threshold as S_l (= 16%), respectively. The TU^D is calculated as 209. The safety bound (f) can be measured using: $f = \frac{(0.21-0.16)\times 209}{1-0.21}$ (= 13.22). Thus, if the total utility in the inserted transactions is below 13.22, it would be redundant to re-scan the initial database since they would definitely not be the HAUUBI in the new database.

4.3 Proposed PRE-HAUIMI algorithm

Before processing the PRE-HAUIMI algorithm, the AUL-structures of 1-HAUUBIs and 1-PAUBBIs are initially constructed in the initial database. With the availability of all AUL-structures in the added transactions, the correctness and completeness hold true for the proposed framework. With the addition of new transactions, the PRE-HAUIMI algorithm analyzes the various cases, and the procedure follows. The pseudo-code for PRE-HAUIMI is given in Algorithm 1.

In Algorithm 1, the buffer (*buf*) is set as 0 by default for the first round (Line 1 in Algorithm 1). The safety bound (f) and the new total utility d are then respectively calculated (Lines 2 to 3 in Algorithm 1). After that, the AUL-structures of all 1-items in d are constructed (Lines 5 to 8 in Algorithm 1). This process is to ensure the correctness and the completeness of the final HAUIs and it is a reasonable process since in real-life situation, there is only a small number of transactions in d compared to the original database D. The constructed AUL-structures in D and d are then merged together by the sub-procedure (Line 9

Algorithm 1 Proposed PRE-HAUIMI.

```
Input: D, the original database; utable, an unit profit table; d,
       insertion transactions; $S_u$, upper-utility threshold; $S_l$,
       lower-utility threshold; D.AULs, the AUL-structures of
       D; $TU^D$, total utility in D.
Output: the set of high average-utility itemsets (HAUIs).
1  set $buf \leftarrow 0$;
2  calculate the safety bound f;
3  calculate total utility in d as $TU^d$;
4  $buf \leftarrow buf + TU^d$;
5  for each $T_q \in d$ do
6      for each $X \in T_q$ do
7          $X.AUL \leftarrow \{T_q, iu, tmu\}$;
8      $d.AULs \leftarrow \cup X.AUL$;
9  Merge(D.AULs, d.AULs, U.AULs);
10 $TU^U := TU^D + TU^d$;
11 for each $X \in U.AULs$ do
12     if $\frac{X.iu.sum}{|X|} \geq TU^U \times S_u$ then
13         $HAUIs \leftarrow \cup X$;
14     if $X.tmu.sum \geq TU^U \times S_u$ then
15         $extAULs \leftarrow null$;
16         for each Y after X in U.AULs do
17             $extAULs \leftarrow extAULs$ + Construct(X.AULs, Y);
18         PRE-HAUIMI(X, extAULs);
19 $D.AULs \leftarrow U.AULs$;
20 return $HAUIs, U.AULs, buf$;
```

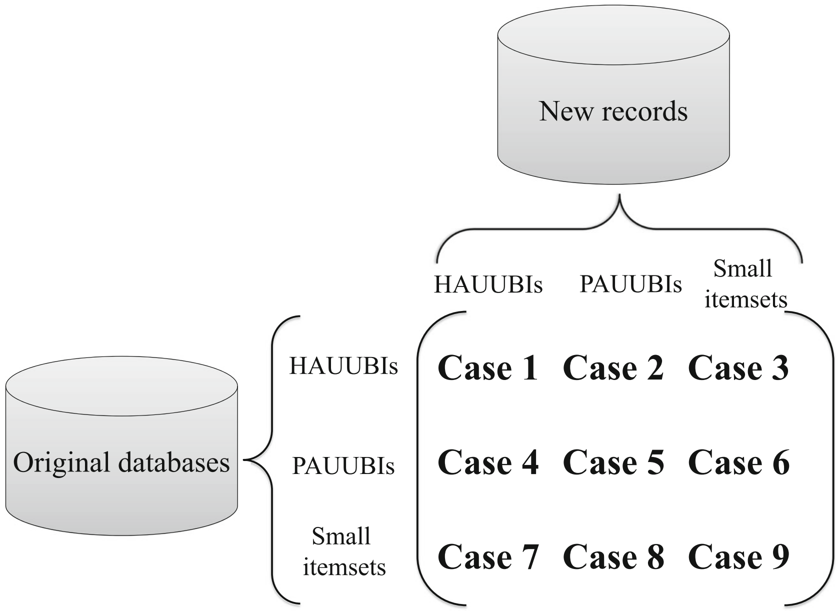


Fig. 4 Nine cases of the utilized pre-large concept

in Algorithm 1). The total utility of the merged databases D and d is then calculated (Line 10 in Algorithm 1). Following this, the updated AUL-structures are then maintained, and if the *auub* value of an itemset X is not larger than the upper-utility, a HAUI is found (Lines 11 to 13 in Algorithm 1), and the supersets of the X verifies the need for scan the supersets (Lines 14 to 17 in Algorithm 1), and the designed PRE-HAUIMI uses recursion for the continuation of the procedure. Then, the updated HAUIs set is maintained and the PHAUIs is set for the buffer (Line 20 in Algorithm 1). Furthermore, the AUL-structures are then updated and kept for the next maintenance (Line 19 in Algorithm 1).

For the **Merge** procedure, if X exists in d and is considered as a HAUUBI, but it does not appear in the AUL-structures of D, the original database D is required to be re-scanned to obtain its AUL-structure (Lines 4 to 6 in Algorithm 2). This approach is to ensure the correctness and completeness of the developed algorithm for the maintenance procedure. In this situation, the *buffer* is then set as 0 (Line 7 in Algorithm 2); otherwise, the *buffer* is updated by the total utility in d. Thus, this value is then used for the next maintenance procedure of the new coming transactions. The AUL-structures of d is then merged and updated with the AUL-structures in D (Lines 10 to 15 in Algorithm 2). In this process, a simple join operation for the new built AUL-structures in d is the performed to directly inserted to the corresponding itemset of AUL-structure in D. After that, the **Construct** function presented in Algorithm 3 verifies the need to scan the supersets considering the *tids* set for combination (Lines 1 to 3 in Algorithm 3). The information of utilities (Lines 4 to 6 in Algorithm 3) are recomputed, and the results form the updated AUL-structures (Lines 7 to 8 in Algorithm 3). This process has a similar complexity to the UL-structure [27] except the pre-large part of 1-PAUUBIs for the PAUL-structure is also built here for maintaining the integrity of the final HAUIs.

Algorithm 2 Merge($D.AULs, d.AULs, U.AULs$).

```
1  set $X.AUL \leftarrow null$;
2  set $U.AULs \leftarrow null$;
3  for each $X \in d.AULs$ do
4      if $\frac{X.tmu.sum}{|X|} \geq TU^d \times S_u \wedge X \notin D.AUL$ then
5          if $TU^d \geq buf$ then
6              scan $D$ to obtain $X.AUL$ from $D$;
7              $buf \leftarrow 0$;
8          else
9              $buf \leftarrow buf + TU^d$;
10     if $\exists X \in D.AULs \wedge X \in d.AULs$ then
11         for each element $E_j \in X.AUL$ do
12             $X.iu.sum \leftarrow X.iu.sum + E_j.iu$;
13             update $X.AUL.tmu$;
14             $X.AUL \leftarrow E_j$;
15         $U.AULs \leftarrow \cup X.AUL$;
```

Algorithm 3 Construct($X.AUL, Y$).

```
   Input: $X.AUL$, the AUL-structures of $X$; $Y$, the itemset $Y$ after
          $X$ in $X.AUL$.
   Output: $XY.AUL$, the AUL-structures of $XY$.
1  $XY.AUL \leftarrow null$;
2  if $\exists E \in Y.AUL \wedge X.AUL.tid == Y.AUL.tid$ then
3      $E_{XY}.AUL.tid \leftarrow X.AUL.tid$;
4      $E_{XY}.iu \leftarrow (X.AUL + Y.AUL)/2$;
5      update $E_{XY}.tmu$;
6      $E_{XY} \leftarrow < E_{XY}.tid, E_{XY}.iu, E_{XY}.tmu >$;
7      $XY.AUL \leftarrow \cup E_{XY}$.
8  return $XY.AUL$;
```

4.4 Complexity analysis

The complexity of the designed PRE-HAUIMI algorithm to maintain the AUL-structure for further mining is analyzed as follows. Again, assume that n is the number of transactions in d , that the number of items in the largest transaction in d is m, it thus requires $O(m \times n)$ time for the first database scan in the worst case. After that, the total utility is calculated by the sum of the transaction utilities in d, which requires $O(n)$. The equation is then used to calculate the safety bound to determine the itemsets, and for each satisfactory case to be examined based on the safety bound, the AUL-structure is then built that requires a linear time $O(m)$ in the worse case if it is implemented as a two-way or three-way search. Assuming that there are k-itemsets of each case in the original database D, a simple operation to join two AUL-structure whether for the original D and the inserted transactions d is $O(k \times N)$ time, where N is the number of the cases for maintenance. Thus, the maintenance part of the AUL-structure in the designed PRE-HAUIMI algorithm is calculated as $O(m \times n + n + m + k \times N)$ at most.

5 A running example

Here, we examine PRE-HAUIMI for incremental maintenance. Before adding the new transactions in the initial database, the AUL-structures of all 1-HAUUBIs and 1-PAUUBIs from the original database were first built. Here, the initial database, the profit table, and the updated transactions are presented in Tables 1, 2, and 3, respectively. The built AUL-structures from the original database was seen in Fig. 2. In this example, let us consider the case where the upper-utility thresshold is configured as S_u (= 21%), and the lower-utility threshold is configured as S_l (= 16%), respectively. The total utility for the initial database and new inserted ones are 209 and 110, and the updated total utility is calculated as (209 + 110) (= 319). In this example, the 1-HAUUBIs and 1-PAUUBIs are shown in Table 4 and their AUL-structures were shown in Fig. 3.

Table 4 The sets of 1-HAUUBIs and 1-PAUUBIs in the original database

Two sets	1-items	*auub*
1-HAUUBIs	*a*	64
	b	58
	c	66
	d	68
1-PAUUBIs	*f*	27

In this example, the *auub* values of all 1-items are given in Table 5.

After all 1-items are obtained from the new transactions forming as the AUL-structures, the elder database and new transactions are then merged together. For instance, the $auub(a)$ is $(64 + 60)(= 124)$. The other 1-items are easily updated since the *auub* values of (b), (c), (d), and (f) were kept in Table 4. In this sample example, the safety bound is calculated as $f = \frac{0.21-0.16}{1-0.21} \times 209$ $(= 13)$. Thus, the original database of the 1-item (e) should be re-scanned to obtain *auub*, and the result of (e) in D is $auub(e)(= 21)$. After the updating process, the 1-HAUUBIs are $\{a, c, d, e, f\}$, and no 1-PAUUBIs in the running example. The enumeration tree of those 1-items are then built and shown in Fig. 2. The supersets of those 1-items are verified for the AUL-structures to be generated using the simple join operation. For example, the generated AUL-structures for 2-itemsets are illustrated in Fig. 5. This procedure is then repeated until all the required HAUIs are then maintained and updated.

From Fig. 5, it is obvious to see that the AUL-structure of (f) is removed in the updated progress. This is reasonable since after the updating progress, the item (f) becomes a small utility pattern, which is then discarded for next generation. Based on the join operation of the *TIDs* (transaction IDs) in the AUL-structures, it is easy to generate the supersets without a huge computational cost; the maintenance performance can be greatly reduced. Moreover, since the complete information of 1-itemsets is kept in the AUL-structure, it is unnecessary to rescan the database for the 2-itemsets based on the downward closure property. Thus, the designed model has better benefit than that of the Apriori-like approach. After all progresses of the designed PRE-HAUIMI, the results of the HAUIs can be then easily obtained.

Table 5 The *auub* values of all 1-items in the new transactions

1-items	auub
a	60
b	4
c	66
d	68
e	60
f	60

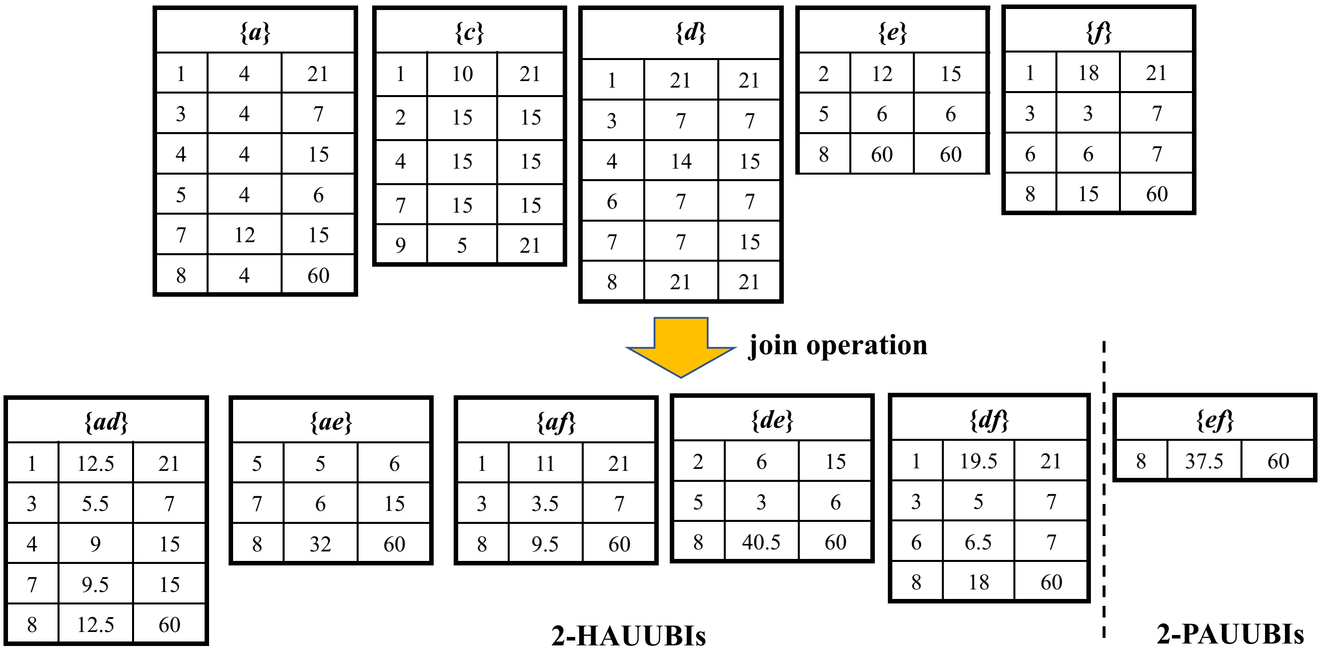
6 Experimentation and findings

Here, we analyze the effectiveness and efficiency of the proposed incremental PRE-HAUIMI algorithm with transaction insertion as compared with FUP-based [36], IHAUPM [33] and the HAUI-Miner [31] algorithms on multiple datasets. The FUP-based ones are considered as the predominant incremental algorithm for updating the discovered HAUIs, and the HAUI-Miner is the predominant approach for the batch progress. All the experiments were carried on Intel(R) Core(TM) i7-6700 4.00GHz processor with 8 GB main memory, with a 64 bits Microsoft Windows 10 OS and implemented in JAVA language. Experimentation was performed on six datasets [9] including four real datasets and two artificial ones, and the artificial datasets were generated using the IBM Quest Synthetic Data Generator [2]. The foodmart dataset contains the quantity and unit profits but for the other datasets, we used synthesis to supplement these values for the datasets. The characteristic of the six datasets used in the experiment are shown in Table 6. The datasets have the following features: the #|**D**| shows the number of transactions; #|**I**| shows the quantity; **AveLen** and **MaxLen** show the average length and maximum length of the transactions, respectively; and **Type** is a Boolean value that represents the datset is whether sparse or dense.

The results were measured for runtime, memory usage, the number of assessed patterns and scalability. Notice that the upper-utility threshold (the same as the high average-utility threshold in traditional HAUIM) is represented as UT for the designed PRE-HAUIMI algorithm. The LT is defined as the lower-utility threshold used in the designed PRE-HAUIMI. The insertion ratio is represented as IR. The IR shows the percentage for the number of inserted transactions from the initial dataset. Notice that the batch model algorithm such as HAUI-Miner should re-scan the updated database to identify the required HAUIs as long as the size of the dataset is updated. The other three incremental algorithms including the developed PRE-HAUIMI only handles the incremental parts for the maintenance since the original dataset was already maintained in the main memory.

6.1 Runtime

Our runtime results for the four algorithms with varying UT and a constant IR (= 1%) are given in Fig. 6. Different



{a}		
1	4	21
3	4	7
4	4	15
5	4	6
7	12	15
8	4	60

{c}		
1	10	21
2	15	15
4	15	15
7	15	15
9	5	21

{d}		
1	21	21
3	7	7
4	14	15
6	7	7
7	7	15
8	21	21

{e}		
2	12	15
5	6	6
8	60	60

{f}		
1	18	21
3	3	7
6	6	7
8	15	60

{ad}		
1	12.5	21
3	5.5	7
4	9	15
7	9.5	15
8	12.5	60

{ae}		
5	5	6
7	6	15
8	32	60

{af}		
1	11	21
3	3.5	7
8	9.5	60

{de}		
2	6	15
5	3	6
8	40.5	60

{df}		
1	19.5	21
3	5	7
6	6.5	7
8	18	60

{ef}		
8	37.5	60

Fig. 5 The AUL-structures of 2-itemsets

datasets may apply different UTs due to the varying specifications for each dataset.

Figure 6 demonstrates that an increase in the upper-bound threshold results in a reduction in the execution time. This can be explained as an increase in the upper-bound value causes the shrinkage of the search space for HAUIs. In most cases, the HAUI-Miner algorithm takes the most computational cost since it performs as the batch model while the size of the transactions is updated with transaction insertion. The IHAUPM uses the tree-based structure, which requires more computational time when compared to the FUP-based approach with list structure. For the designed algorithm, it takes less computational cost for the incremental mining, since it relies on the efficient AUL-structure and the pre-large concept. For cases with a smaller total utility in the initial database than the set safety bound, there is no need to scan initial database again but the completeness and correctness of the discovered HAUIs can be easily maintained. The results for changing IRs and a constant UT are presented in Fig. 7.

From Fig. 7, all the algorithms remain stable with the increasing of the IR. The reason is that the IR slightly affects the mining performance since most rules were discovered. The proposed algorithm still has best performance since re-scans are redundant. Results of the memory usage for all algorithms are also presented.

6.2 Memory usage

Our experimentation included an analysis of memory usage with changing UTs and a constant IR (Fig. 8).

Figure 8 demonstrates that the IHAUPM requires more memory as compared to other approaches for most scenarios, e.g. in Fig. 8a, b, d, e, and f. The reason is

Table 6 Dataset specifications

Dataset	#|**D**|	#|**I**|	AvgLen	MaxLen	Type
retail	88,162	16,407	10	76	Sparse
T10I4D100K	100,000	870	10.1	29	Sparse
kosarak	990,002	41,270	8	2,498	Sparse
T40I10D100K	100,000	1,000	39.6	77	Dense
mushroom	963	119	23	23	Dense
foodmart	21,556	1,559	4	11	Sparse

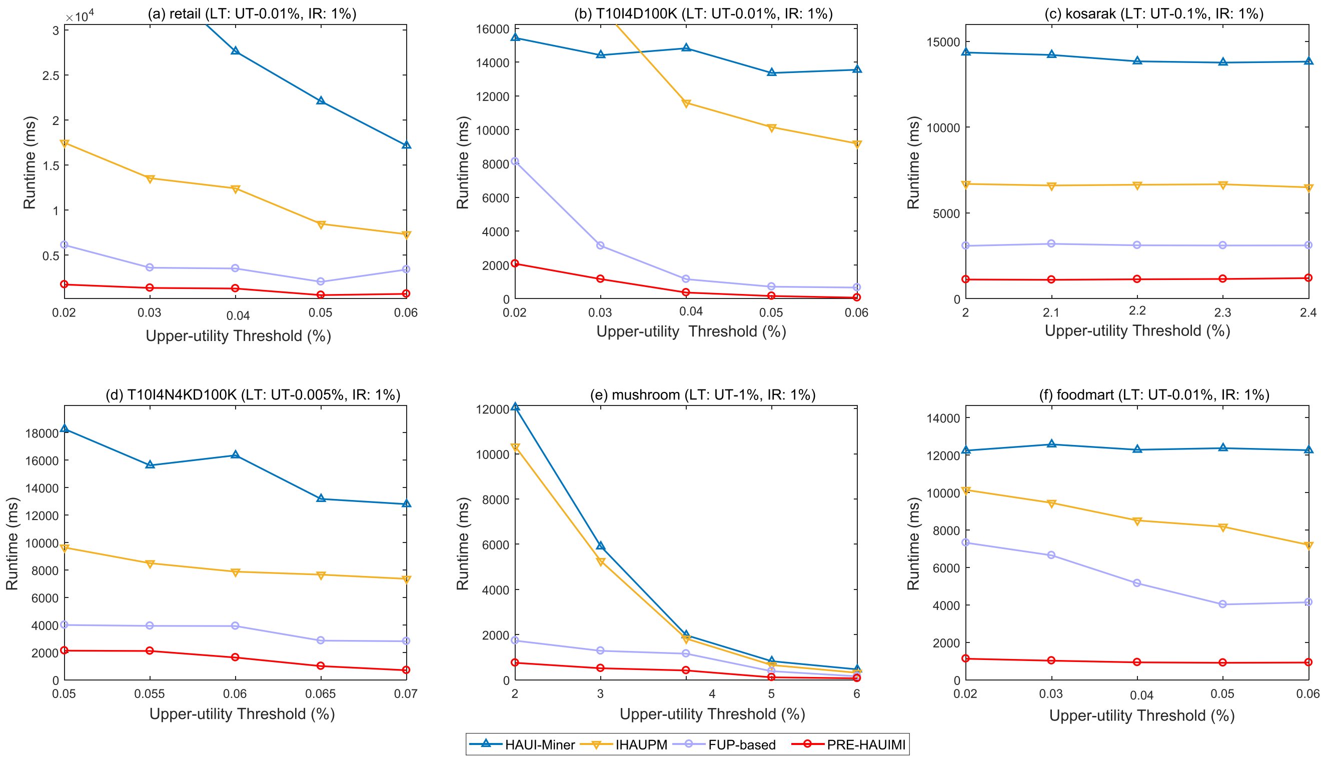


Fig. 6 Runtime results w.r.t. varying thresholds

that the IHAUPM uses the tree structure as a buffer; it is efficient but still has worse performance than that of the AUL-structure. The proposed PRE-HAUIMI has better results than that of the FUP-based algorithm since there is no need to re-scan the initial database, but sometimes needs more memory usage to keep the extra information

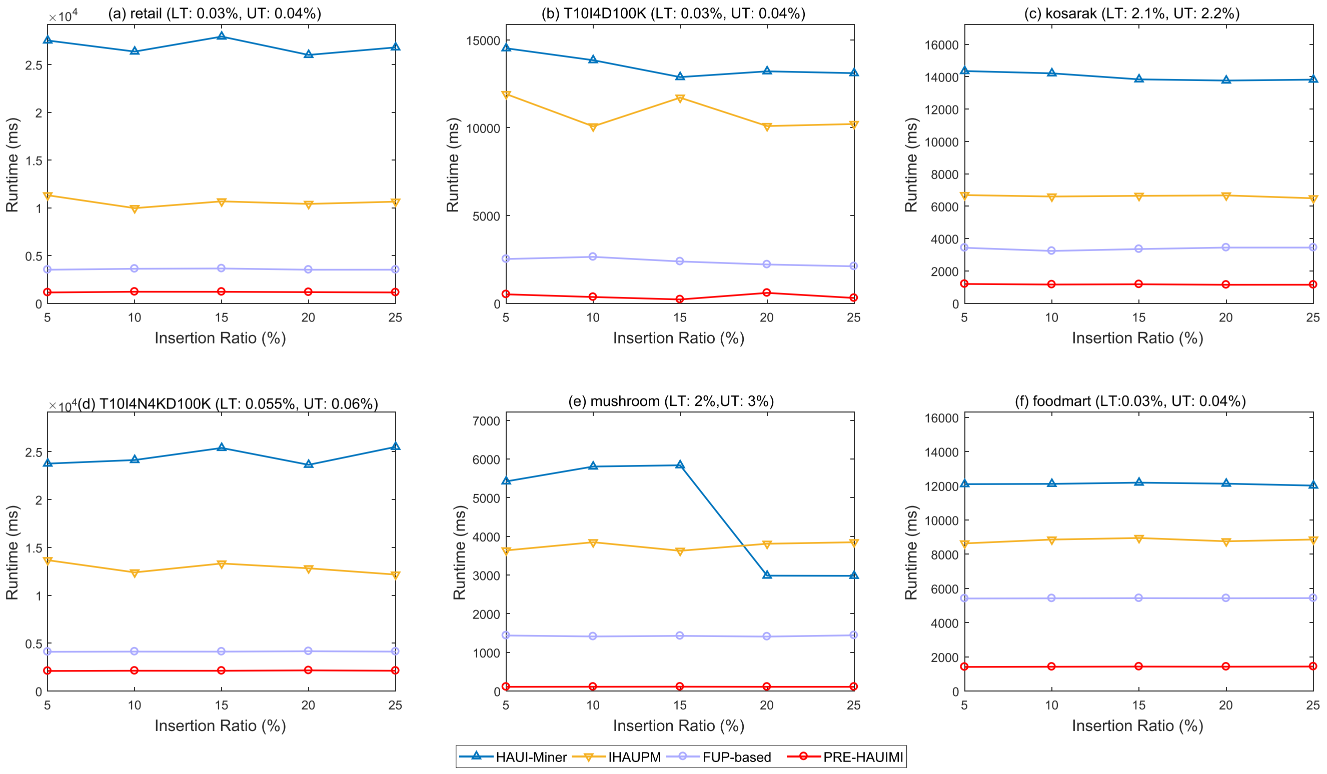


Fig. 7 The results of runtime w.r.t. varied insertion ratios

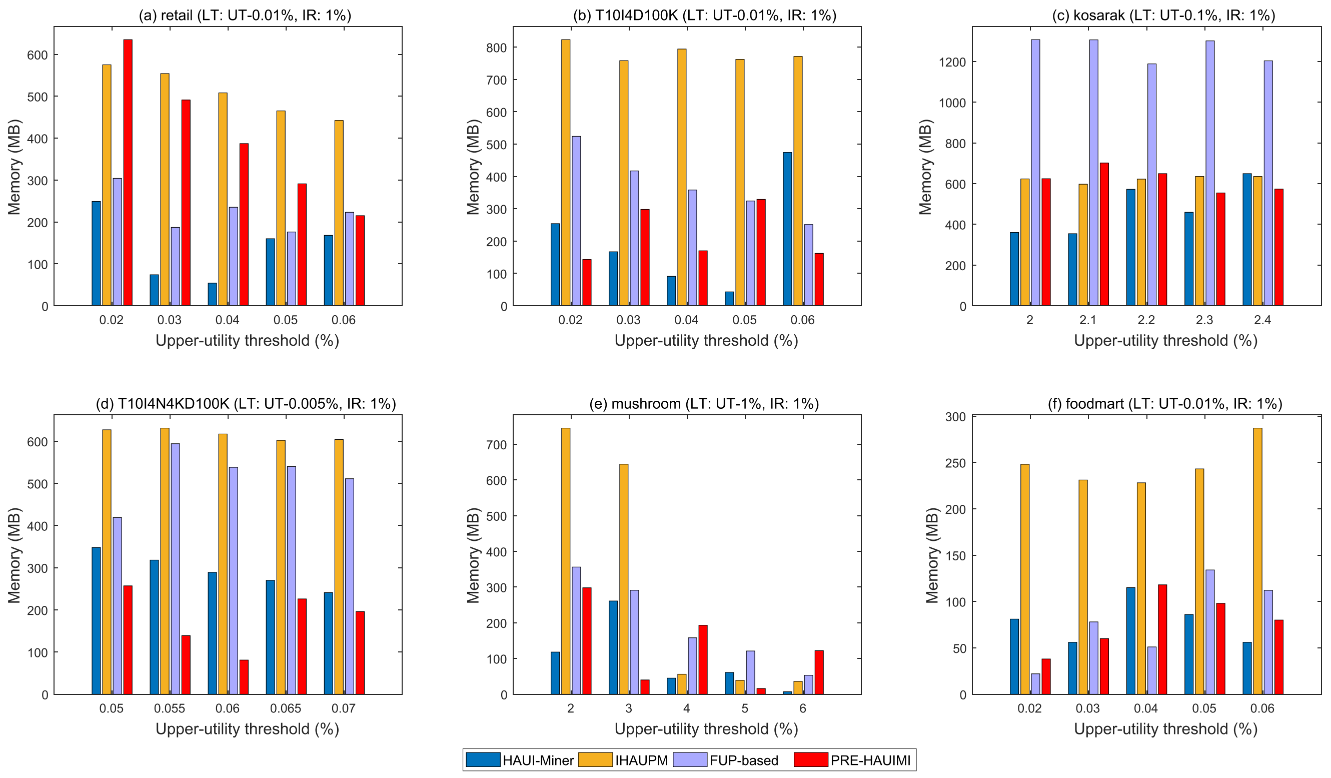


Fig. 8 Memory usage results w.r.t changing thresholds

(pre-large utility patterns), which can be found in Fig. 8a when the upper-bound threshold is set lower. The HAUI-Miner needs less memory usage in most cases due to its batch-mode dependency on a change in data size; storing the extra information for later mining progress is redundant, and results in a high computational cost. For the incremental

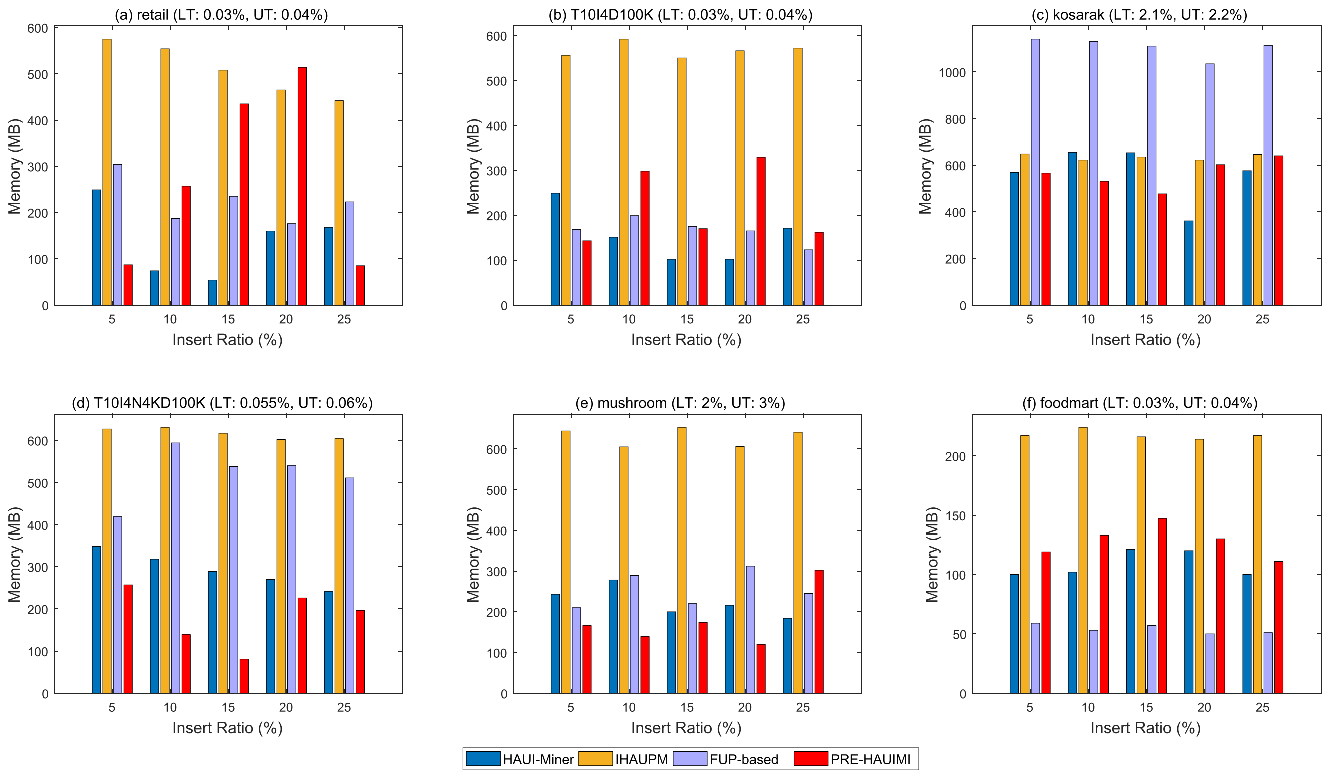


Fig. 9 Memory usage results w.r.t changing insertion ratios

progress, generally, the designed PRE-HAUIMI performs well in terms of memory needs in most datasets. Results in terms of varied IRs with a constant UT are given in Fig. 9.

Figure 9 demonstrates similar results to Fig. 8; i.e. the HAUI-Miner has a lower memory requirement, and the IHAUPM needs the most memory usage for incremental mining in most cases except in kosarak dataset. This is interpreted as the AUL-structure performs well in incremental scenarios. Although the PRE-HAUIMI requires more memory to buffer the required information of the pre-large utility patterns than that of the FUP-based approach, for example in Fig. 9a, b, d, and e, but it is a trade-off problem between the runtime and the memory usage. Only a very slight extra memory is required for the developed PRE-HAUIMI algorithm than that of the FUP-based approach but still outperforms the IHAUPM algorithm.

6.3 Number of assessed patterns

Here, we analyze our results for the number of assessed patterns to find the HAUIs. The results for changing UTs and a constant IR are presented in Figs. 10 and 11.

From Figs. 10 and 11, we can see that the assessed patterns of the developed PRE-HAUIMI is much fewer than the other algorithms except in Figs. 10c and 11c. From the Table 6, we can observe the sparsity of the kosarak dataset with a very high the maximal length in some transactions. Thus, the processed inserted transactions for the incremental mining may consists of long lengths; the developed PRE-HAUIMI may keep more information as the pre-large utility patterns for later mining process. In most cases, the IHAUPM needs to asses more patterns since it adopts the tree structure.

6.4 Scalability

Here, we perform experimentation on the scalability of the different appoaches using artificial datasets, synthesizes by IBM Generator [2]. With a varying ratio of scalability from 100*K* to 500*K*, in 100*K* increments. These measurements are represented in Fig. 12.

Figure 12 shows that PRE-HAUIMI has better performance in terms of runtime and the number of assessed patterns. For memory usage, since the developed PRE-HAUIMI needs to maintain the pre-large utility patterns, it sometimes needs extra memory to keep those information. The IHAUPM needs the most memory usage than the other algorithms in terms of memory usage. This can be explained as HAUI-Miner, FUP-based and the developed PRE-HAUIMI adopt the efficient AUL-structure, which requires less memory usage than the tree-based algorithm. We also can see that the proposed PRE-HAUIMI is more stable as compared to others in terms of runtime, which

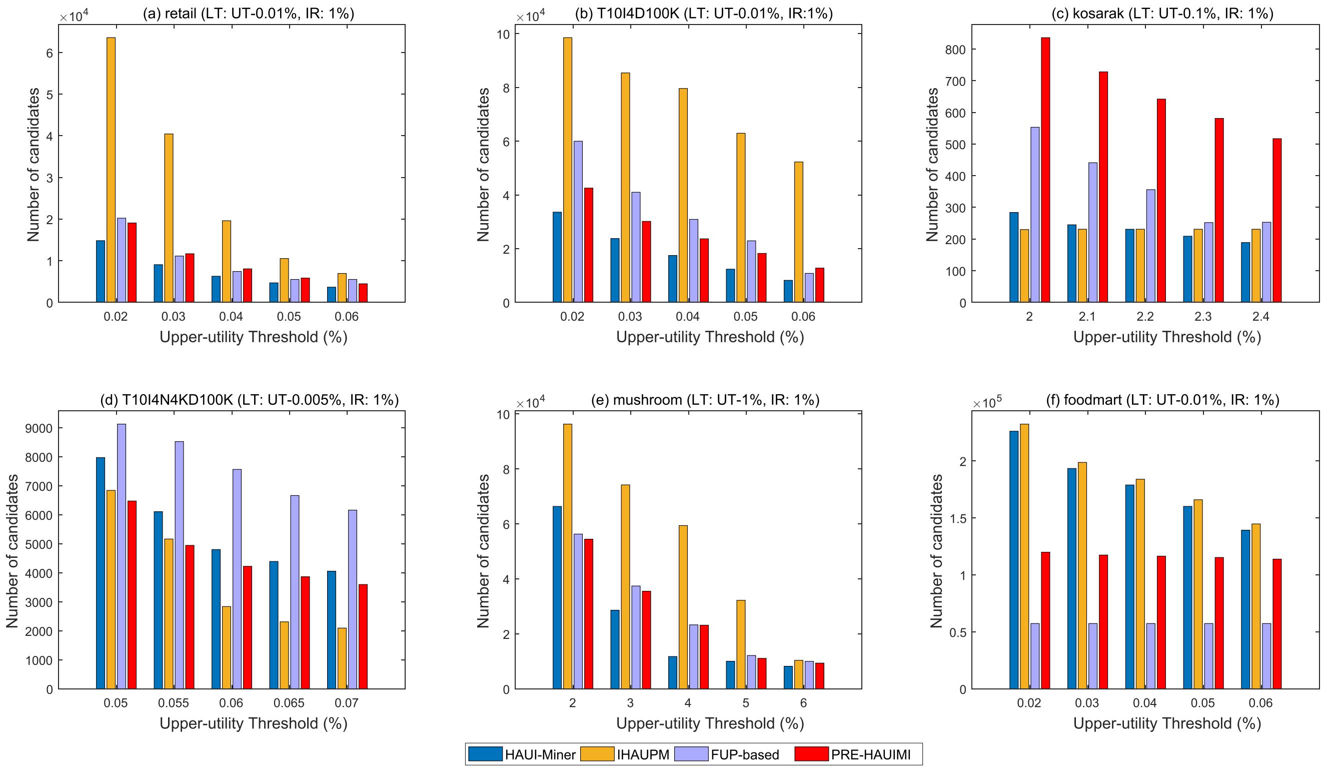


Fig. 10 The results of number of assessed patterns w.r.t varied thresholds

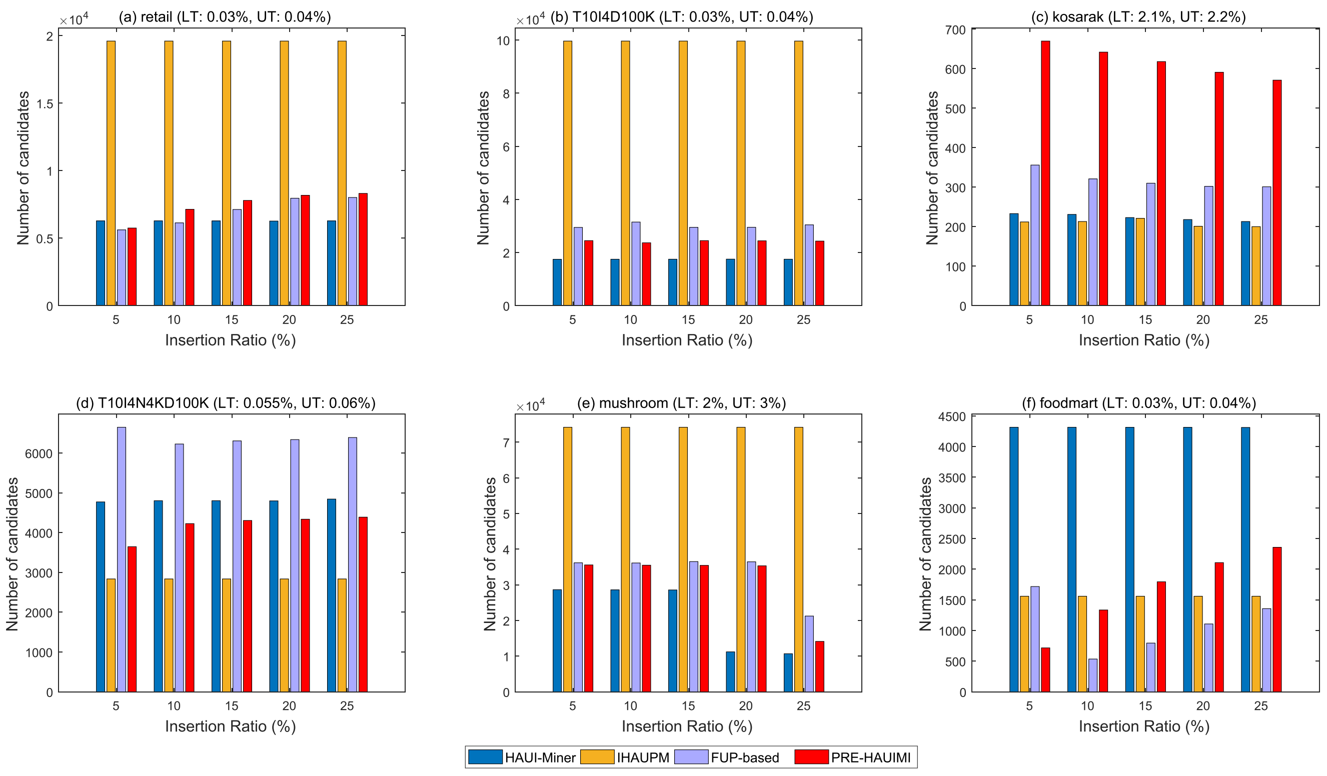


Fig. 11 The results of number of assessed patterns w.r.t insertion ratios

shows better efficiency and effectiveness in the updating progress. From the results, it also indicated that the performance is not stable according to the size of the increasing database. This is reasonable since the designed PRE-HAUIMI needs to keep more itemsets in the pre-large concept, it needs to explore more candidates for maintenance rather than the FUP-based model, which can be seen in Fig. 12c (dataset size is 100K or 200K). Moreover, since the designed algorithm keeps an extra AU-structures based on the pre-large concept, it sometimes needs more memory usage than that of the FUP-based model that can be seen in Fig. 12b (dataset size is 500K). In average, the designed algorithm has less computational cost, memory usage, and number of determined candidates. Thus, we can conclude that the designed PRE-HAUIMI can well handle the incremental progress for updating the discovered HAUIs.

7 Conclusion and future work

With the prevalence of High-average utility mining (HAUIM) because of its fair scale in evaluating the average utilities of itemsets as compared to the traditional high utility itemset mining (HUIM). Most existing algorithms of HAUIM handle the static database; with updates in the

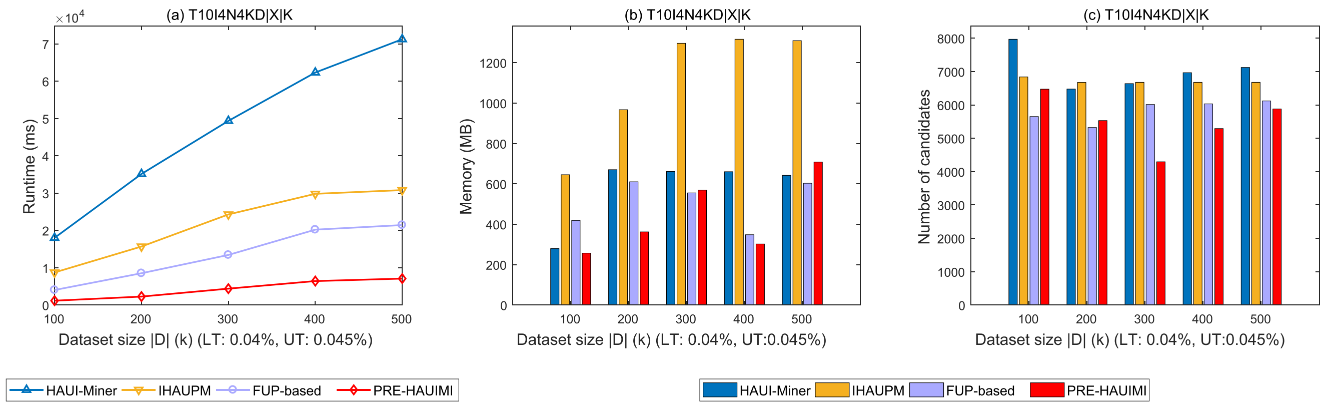


Fig. 12 Scalability under varied database size

dataset, the new database has to go through redundant re-scans to be able to find the updated patterns. Thus, the previous findings go to waste, which has a high overhead in terms of computational cost and memory needs. In this article, we developed and experimented the PRE-HAUIMI algorithm, which addresses the incremental mining problem with transaction insertion. A pre-large concept is utilized here for HAUIM, and mining is sped up using the AUL-structure. With the design of a safety bound, the database re-scans are prevented in the maintenance procedure but the correctness and completeness can still be held. The results demonstrate the out-performance of the PRE-HAUIMI algorithm over the traditional HAUI-Miner algorithm running in the batch model and the IHAUPM and FUP-based approaches running in the incremental maintenance.

There is still a limitation of the designed PRE-HAUIMI algorithm. For example, if the number of processed itemsets of each case is huge in the newly inserted transaction, then it takes time to build the AUL-structure for the itemsets in the new transactions. This issue has a major impact on the computational cost. This situation happens when a transaction contains many items in the newly inserted transactions especially for the sales periods or seasons (i.e., Christmas or black Friday sales). The solution for this issue is to integrate the original database and new transactions as an updated database, then applying the state-of-the-art HAUIM algorithm to mine the required HAUIs. This is reasonable since the most up-to-date HAUIM algorithm applies the efficient pruning strategies to quickly reduce the size of the unpromising candidates. Although the designed PRE-HAUIMI algorithm cannot obtain the benefits from the above situation, for the real-life applications especially for the basket-market analysis, the developed PRE-HAUIMI algorithm can well handle the dynamic database since the size of the daily inserted transactions and the itemsets within the transactions is much less than that of the historical database.

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

References

1. Agrawal R, Srikant R (1994) Fast algorithms for mining association rules in large databases. The International Conference on Very Large Data Bases, pp 487–499
2. Agrawal R, Srikant R (1994) Quest synthetic data generator. http://www.Almaden.ibm.com/cs/quest/syndata.html
3. Ahmed CF, Tanbeer SK, Jeong BS, Lee YK (2009) Efficient tree structures for high utility pattern mining in incremental databases. IEEE Trans Knowl Data Eng 21(12):1708–1721
4. Erwin A, Gopalan RP, Achuthan NR (2008) Efficient mining of high utility itemsets from large datasets. The Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining, pp 554–561
5. Cheung DW, Wong CY, Han J, Ng VT (1996) Maintenance of discovered association rules in large databases: an incremental updating techniques. The International Conference on Data Engineering, pp 106–114
6. Chen MS, Park JS, Yu PS (1998) Efficient data mining for path traversal patterns. IEEE Trans Knowledge Data Eng 10(2):209–221
7. Creighton C, Hanash S (2003) Mining gene expression databases for association rules. Bioinformatics 19(1):79–86
8. Deng Z, Lv SL (2014) Fast mining frequent itemsets using nodesets. Expert Syst Appl 41(10):4505–4512
9. Fournier-Viger P, Lin JCW, Gomariz A, Gueniche T, Soltani A, Deng Z, Lam HT (2016) The SPMF open-source data mining library version 2. Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pp 36–40
10. Fournier-Viger P, Zida S, Lin JCW, Wu CW, Tseng VS (2016) EFIM-closed: fast and memory efficient discovery of closed high-utility itemsets. International Conference on Machine Learning and Data Mining in Pattern Recognition, pp 199–213
11. Fournier-Viger P, Li Z, Lin JCW, Kira RU, Fujita H (2019) Efficient algorithms to identify periodic patterns in multiple sequences. Inform Sci 489:205–226
12. Gan W, Lin JCW, Chao HC, Fujita H, Yu PS (2019) Correlated utility-based pattern mining. Inf Sci 504:470–486
13. Le NT, Vo B, Nguyen LBQ, Fujita H, Le B (2020) Mining weighted subgraphs in a single large graph. Inf Sci 514:149–165
14. Hong TP, Wang CY, Tao YH (2001) A new incremental data mining algorithm using pre-large itemsets. Intelligence Data Analysis 5:111–129
15. 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
16. Hong TP, Lin CW, Wu YL (2008) Incrementally fast updated frequent pattern trees. Expert Syst Appl 34(4):2424–2435
17. Hong TP, Lee CH, Wang SL (2011) Effective utility mining with the measure of average utility. Expert Syst Appl 38(7):8259–8265
18. Krishnamoorthy S (2019) A comparative study of top-K high utility itemset mining methods. High-Utility Pattern Mining, pp 47–74
19. Liu Y, Liao WK, Choudhary A (2005) A fast high utility itemsets mining algorithm. The International Workshop on Utility-Based Data Mining, pp 90–99
20. Liu Y, Liao WK, Choudhary A (2005) A two-phase algorithm for fast discovery of high utility itemsets. The Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining, pp 689–695
21. Lucchese C, Orlando S, Perego R (2006) Fast and memory efficient mining of frequent closed itemsets. IEEE Transactions of Knowledge and Data Engineering 18(1):21–36

22. Lin CW, Lan GC, Hong TP (2009) An incremental mining algorithm for high utility itemsets. Expert Syst Appl 39(8):7173–7180
23. Lin CW, Hong TP, Lu WH (2010) Maintaining high utility pattern trees in dynamic databases. The International Conference on Computer Engineering and Applications. pp, 304–308
24. Lin CW, Hong TP, Lu WH (2010) Efficiently mining high average utility itemsets with a tree structure. The Asian Conference on Intelligent Information and Database Systems, pp 131–139
25. Lin CW, Hong TP, Lu WH (2011) An effective tree structure for mining high utility itemsets. Expert Syst Appl 38(6):7419–7424
26. Liu J, Wang K, Fung BCM (2012) Direct discovery of high utility itemsets without candidate generation. IEEE International Conference on Data Mining, pp 984–989
27. Liu M, Qu J (2012) Mining high utility itemsets without candidate generation. ACM International Conference on Information and Knowledge Management, pp 55–64
28. Lan GC, Hong TP, Tseng VS (2012) A projection-based approach for discovering high average-utility itemsets. J Inf Sci Eng 28:193–209
29. Lu T, Vo B, Nguyen HT, Hong TP (2014) A new method for mining high average utility itemsets. Computer Information Systems and Industrial Management, pp 33–42
30. Liu J, Wang K, Fung BCM (2016) Mining high utility patterns in one phase without generating candidates. IEEE Trans Knowl Data Eng 28(5):1245–1257
31. Lin CW, Li T, Fournier-Viger P, Hong TP, Zhan J, Voznak M (2016) An efficient algorithm to mine high average-utility itemsets. Adv Eng Inform 30(2):233–243
32. Lin JCW, Ren S, Fournier-Viger P, Hong TP (2017) EHAUPM: efficient high average-utility pattern mining with tighter upper-bound models. IEEE Access 5:12927–12940
33. Lin JCW, Ren S, Fournier-Viger P, Pan JS, Hong TP (2018) Efficiently updating the discovered high average-utility itemsets with transaction insertion. Eng Appl Artif Intell 72:136–149
34. Nguyen LTT, Vu VV, Lam MTH, Duong TTM, Manh LT, Nguyen TTT, Vo B, Fujita H (2019) An efficient method for mining high utility closed itemsets. Inf Sci 495:78–99
35. Tseng VS, Shie BE, Wu CW, Yu PS (2013) Efficient algorithms for mining high utility itemsets from transactional databases. IEEE Trans Knowl Data Eng 25(8):1772–1786
36. Wu TY, Lin JCW, Shao Y, Fournier-Viger P, Hong TP (2018) Updating the discovered high average-utility patterns with transaction insertion. The International Conference on Genetic and Evolutionary Computing, pp 66–73
37. Wu JMT, Teng Q, Lin JCW, Yun U, Chen HC (2020) Updating high average-utility itemsets with pre-large concept. Journal of Intelligent & Fuzzy Systems 2020:1–10
38. Yao H, Hamilton HJ, Butz CJ (2004) A foundational approach to mining itemset utilities from databases. SIAM International Conference on Data Mining, pp 215–221
39. Yen SJ, Lee YS (2007) Mining high utility quantitative association rules. The International Conference on Data Warehousing and Knowledge Discovery, pp 283–292
40. Yun U, Kim D (2017) Mining of high average-utility itemsets using novel list structure and pruning strategy. Future Generation Computer System 68:346–360
41. Yun U, Kim D, Yoon E, Fujita H (2018) Damped window based high average utility pattern mining over data streams. Knowl-Based Syst 144:188–205
42. 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 51:595–625

Jerry Chun-Wei 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 300 research articles in refereed journals (IEEE TKDE, IEEE TCYB, IEEE TII, IEEE SysJ, IEEE SensJ, ACM TKDD, ACM TDS, ACM TMIS) and international conferences (IEEE ICDE, IEEE ICDM, PKDD, PAKDD). His research interests include data mining, soft computing, artificial intelligence and 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 of IEEE TII, ACM TMIS, IEEE Access, JIT, PlosOne, IDA, and IJIMAI. He is the Fellow of IET (FIET), senior member for both IEEE and ACM.

Matin Pirouz is a faculty member with the Department of Computer Science at California State University, Fresno. Her current research interests include CS Education, big data analytics, social network analysis, and data mining. Her current projects include applying prescriptive and descriptive analyses. Her research has been funded by the National Science Foundation and other federal, state, and private funding agencies.

Youcef Djenouri obtained the PhD in Computer Engineering from the University of Science and Technology USTHB Algiers, Algeria, in 2014. He was granted a post-doctoral fellowship from the Unist university on South Korea, and he worked on BPM project supported by Unist university in 2016. In 2017, he was post-doctoral research at Southern Denmark University, where he has working on urban traffic data analysis. He was granted a post-doctoral fellowship from the European Research Consortium on Informatics and Mathematics (ERCIM), and worked at the Norwegian University of Science and Technology (NTNU), in Trondheim, Norway. He is working as the researcher in SINTEF Digital, Oslo, Norway. He is working on topics related to artificial intelligence and data mining, with focus on association rules mining, frequent itemsets mining, parallel computing, swarm.

Chien-Fu Cheng received the Ph.D. degree in computer science from National Chiao-Tung University, Hsinchu, Taiwan, in 2008. He is currently a full professor with the Department of Computer Science and Information Engineering and Graduate Institute of Networking & Multimedia, Tamkang University, New Taipei, Taiwan. Dr. Cheng published many papers in several prestigious conferences and journals, such as IEEE LCN, IEEE MASS, IEEE VTC, IEEE WCNC, IEEE Trans. Mobile Comput., IEEE Trans. Veh. Technol., IEEE Internet Things J., IEEE Sensors J., and IEEE Commun. Lett. His current research interests include wireless communication and mobile computing, wireless ad hoc and sensor networks, distributed computing and fault tolerant computing. Dr. Cheng served as a TPC Member of more than 50 communication and computer conferences, such as IEEE GLOBECOM, IEEE ICC, IEEE INFOCOM, and IEEE WCNC. He is currently an associate editor for the IEEE Access. For more details, please refer to his website (http://mail.tku.edu.tw/cfcheng/eng.html).

Usman Ahmed is a PhD candidate at the Western Norway University of Applied Sciences (HVL). He has rich experience in building and scaling high-performance systems based on data mining, natural language processing, and machine learning. His research interests are sequential data mining, heterogeneous computing, natural language processing, recommendation systems, and machine learning.

Affiliations

Jerry Chun-Wei Lin[1] · **Matin Pirouz**[2] · **Youcef Djenouri**[3] · **Chien-Fu Cheng**[4] · **Usman Ahmed**[1]

Matin Pirouz
mpirouz@ieee.org

Youcef Djenouri
youcef.djenouri@sintef.nom

Chien-Fu Cheng
cfcheng@mail.tku.edu.tw

Usman Ahmed
Usman.Ahmed@hvl.no

1 Department of Computer Science, Electrical Engineering and Mathematical Sciences, Western Norway University of Applied Sciences, Bergen, Norway

2 Department of Computer Science, California State University, Fresno, CA, USA

3 SINTEF Digital, Mathematics and Cybernetics, Oslo, Norway

4 Department of Computer Science and Information Engineering, Tamkang University, New Taipei City, Taiwan