Knowledge and Information Systems (2021) 63:2947–2967
https://doi.org/10.1007/s10115-021-01614-z

REGULAR PAPER

Generalized maximal utility for mining high average-utility itemsets

Wei Song[1,2] · Lu Liu[1] · Chaomin Huang[1]

Received: 17 January 2019 / Revised: 18 September 2021 / Accepted: 2 October 2021 /
Published online: 21 October 2021
© The Author(s), under exclusive licence to Springer-Verlag London Ltd., part of Springer Nature 2021

Abstract
Mining high average-utility itemsets (HAUIs) is a promising research topic in data mining because, in contrast to high utility itemsets, they are not biased toward long itemsets. Regardless of what upper bounds and pruning strategies are used, most existing HAUI mining algorithms are founded on the concept of maximal utility, namely the highest utility of a single item in each transaction. In this paper, we study this problem by generalizing the typical maximal utility and average-utility upper bound from a single item to an itemset, and propose an efficient HAIU mining algorithm based on generalized maximal utility (HAUIM-GMU). For this algorithm, we first propose the concepts of generalized maximal utility and the generalized average-utility upper bound, and discuss how the proposed upper bound can be made tighter to generate fewer candidates. A new pruning strategy is then proposed based on the concept of support, and this is shown to be effective for filtering out unpromising itemsets. The final algorithm is described in detail. Extensive experimental results show that the HAUIM-GMU algorithm outperforms existing state-of-the-art algorithms.

Keywords Data mining · High average-utility itemset · Generalized maximal utility · Generalized average-utility upper bound · Critical support count

1 Introduction

One major topic of data mining is frequent itemset (FI) mining [1,21]. The aim of FI mining is to discover sets of items that occur with high frequencies in transaction databases. However, the major frequency metric in FI does not reflect the profit or utility of different items. As an extension of FI, high utility itemset (HUI) mining [16,19] mines itemsets with high profits by considering both the quantity and value of a single item. Recently, the problem of mining HUIs has received considerable attention, and several algorithms [2,6,20,22] have been proposed.

✉ Wei Song
songwei@ncut.edu.cn

1 School of Information Science and Technology, North China University of Technology, Beijing 100144, China

2 Beijing Key Laboratory on Integration and Analysis of Large-scale Stream Data, Beijing 100144, China

Although HUI mining is better at discovering profitable itemsets than FI mining, which only considers frequency, it is obvious that longer itemsets tend to have higher utilities. Using the same utility threshold, itemsets with fewer items are less likely to be discovered. To balance the profit and length of the itemsets, the problem of high average-utility itemset mining (HAUIM) was proposed [5]. In contrast to HUIs, high average-utility itemsets (HAUIs) normalize the utility of the itemsets according to the length. Thus, itemsets with higher unit profits can be discovered.

The original HAUIM algorithm traverses the search space in a breadth-first manner, using the maximal utility of a single item in each transaction as an upper bound [5]. As in early itemset mining algorithms using a level-wise candidate generation-and-test roadmap, this algorithm suffers from the problem of several database scans and numerous candidate itemsets. Several HAUIM algorithms have subsequently been proposed. Different data structures [10,12] and different pruning strategies [9,25] are used in these algorithms. Other problems such as top-k HAUIM [24], HAUIM with multiple thresholds [11,13], HAUIM in dynamic databases [7,15], HAUIM from data streams [26,27], and parallel HAUIM [18] have also been studied. Among these algorithms, reducing the number of candidates is the main consideration. Thus, designing a tighter upper bound [8,14] is a key factor of the HAUIM problem.

To deal with the problem of HAUIM, our motivation and algorithmic considerations are as follows. In existing algorithms, whether maximal utility, remaining maximal utility, or revised transaction-maximum utility, the basic component of the different upper bounds is always the utility value of single items. Thus, the first algorithmic consideration is generalizing the object of existing upper bounds from an item to an itemset to provide a more precise approximation and produce fewer candidates. Furthermore, to the best of our knowledge, the support, which is the main parameter for mining FIs, has not been used for mining HAUIs. Thus, we propose a new pruning strategy from the perspective of the support. The calculation of the support occurs within the first database scan, so no further operations are needed.

In this paper, we propose a new generalized average-utility upper bound (GAUUB) to overestimate the real average-utility. We prove that the GAUUB is tighter than the typical average-utility upper bound. To traverse the search space efficiently, a support-based pruning strategy is also presented. Using the new upper bound model and pruning strategy, a novel algorithm called High Average-Utility Itemset Mining Based on Generalized Maximal Utility (HAUIM-GMU) is proposed. The experimental results demonstrate the superiority of HAUIM-GMU over several existing methods.

The remainder of this paper is organized as follows. We describe the problem of HAUIM in Sect. 2. Section 3 then reviews some related work. The proposed algorithm is explained in Sect. 4. Experimental results are presented and analyzed in Sect. 5. Finally, we draw our conclusions in Sect. 6.

2 HAUIM problem

Let $\boldsymbol{I} = \{i_1, i_2, \ldots, i_M\}$ be a finite set of items. Then, the set $X \subseteq \boldsymbol{I}$ is called an *itemset*; an itemset containing k items is called a k-itemset. Let $\boldsymbol{D} = \{T_1, T_2, \ldots, T_N\}$ be a transaction database. Each transaction $T_i \in \boldsymbol{D}$, with the unique identifier Tid, is a subset of $\boldsymbol{I}$.

The *internal utility* $q(i_p, T_d)$ represents the quantity of item i_p in transaction T_d. The *external utility* $p(i_p)$ is the unit profit value of item i_p. The $utility$ of item i_p in transaction T_d is defined as $u(i_p, T_d) = p(i_p) \times q(i_p, T_d)$. The utility of itemset X in transaction T_d is defined as $u(X, T_d) = \sum_{i_p \in X \wedge X \subseteq T_d} u(i_p, T_d)$. The utility of itemset X in $\boldsymbol{D}$ is defined as

Table 1 Example database

Tid	Transactions	*mu*
T_1	$(A, 1)(B, 1)(C, 1)(F, 2)$	8
T_2	$(B, 1)(D, 1)(E, 1)$	6
T_3	$(A, 1)(B, 1)(C, 1)(F, 1)$	7
T_4	$(C, 3)(D, 2)(F, 1)$	9
T_5	$(A, 1)(C, 2)$	7

Table 2 Profit table

Item	*A*	*B*	*C*	*D*	*E*	*F*
Profit	7	1	3	2	6	4

$u(X) = \sum_{X \subseteq T_d \wedge T_d \in \boldsymbol{D}} u(X, T_d)$. The *transaction utility* (TU) of transaction T_d is defined as $TU(T_d) = u(T_d, T_d)$.

The *average-utility* of a k-itemset X in transaction T_d is defined as $au(X, T_d) = u(X, T_d)/k$. The average-utility of X in transaction database $\boldsymbol{D}$ is defined as $au(X) = \sum_{X \subseteq T_d \wedge T_d \in \boldsymbol{D}} au(X, T_d)$.

The *minimum average-utility threshold* δ, specified by the user, is defined as a percentage of the total TU values of the database, whereas the *minimum average-utility value* is defined as $min\_util = \delta \times \sum_{T_d \in \boldsymbol{D}} TU(T_d)$. An itemset X is called an HAUI if $au(X) \geq min\_util$; otherwise, X is called a low average-utility itemset (LAUI). An HAUI/LAUI with k items is called a k-HAUI/k-LAUI. Given a transaction database $\boldsymbol{D}$, the task of HAUIM is to determine all itemsets that have average-utilities no less than min_util.

The *maximal utility* of transaction $T_d \in \boldsymbol{D}$ is defined as $mu(T_d) = max\{u(i_j) \mid i_j \in T_d\}$. The *average-utility upper bound* (AUUB) of itemset X is defined as $auub(X) = \sum_{X \subseteq T_d} mu(T_d)$. If $auub(X) \geq min\_util$, X is called a *high AUUB itemset* (HAUUBI). An HAUUBI with k items is called a k-HAUUBI. The AUUB model is generally used as a downward closure property to improve the mining efficiency of HAUIs [5,12].

Consider the transaction database in Table 1 and the profit table in Table 2. For convenience, we denote an itemset $\{C, F\}$ as CF. In the example database, the utility of item C in transaction T_1 is $u(C, T_1) = 3 \times 1 = 3$, the utility of itemset CF in transaction T_1 is $u(CF, T_1) = u(C, T_1) + u(F, T_1) = 3 + 8 = 11$, and the utility of itemset CF in the transaction database is $u(CF) = u(CF, T_1) + u(CF, T_3) + u(CF, T_4) = 11 + 7 + 13 = 31$. The average-utility of CF in the transaction database is $au(CF) = (u(CF, T_1) + u(CF, T_3) + u(CF, T_4))/2 = 15.5$. Given $min\_util = 16$, we have $au(CF) < min\_util$, and CF is thus not an HAUI. However, when using the same threshold, CF is an HUI because $u(CF) > min\_util$. The maximal utility of T_1 is $mu(T_1) = max\{u(A, T_1), u(B, T_1), u(C, T_1), u(F, T_1)\} = 8$. The maximal utilities of the other transactions are listed in the third column of Table 1. The AUUB of CF is $auub(CF) = mu(T_1) + mu(T_3) + mu(T_4) = 24$. Because $auub(CF) > min\_util$, CF is an HAUUBI.

3 Existing algorithms

The problem of HAUIM was first proposed by Hong et al. [5]. Similar to the two-phase method [16] for mining HUIs, an HAUIM algorithm called TPAU that uses level-wise traversal of the HAUI search space was proposed. Furthermore, the AUUB was proposed to approximate the average-utility of an itemset. This breadth-first algorithm suffers from the problem of multiple database scans, much like the apriori algorithm [1] for mining FIs.

Subsequently, a tree-based algorithm called HAUP-growth [10], which follows the pattern-growth methodology [4], was proposed. Similar to other tree-based pattern mining algorithms, HAUP-growth first constructs an HAUP tree, then generates HAUUBIs from the tree using the pattern-growth approach, and finally identifies the HAUIs from the set of candidates. Although the tree structure is often compact, it may not be minimal and still occupies a large memory space.

Lan et al. proposed a projection-based HAUIM algorithm called PBAU [9]. Instead of building a tree structure, PBAU uses a prefix-based projection to avoid unpromising itemsets. Furthermore, an index table is used for efficient computation by linking the transactions of the processing itemsets. Lu et al. [17] proposed the HAUI-Tree algorithm for mining HAUIs. To facilitate the mining efficiency, for a node representing itemset X, $au(X)$, $auub(X)$, the set of transaction identifiers containing X, and the last positions of X in transactions containing it are stored.

A list is another data structure for efficient HAUIM. An average-utility (AU)-list was first used in the HAUI-Miner algorithm [12]. For each itemset, the transaction identifier, utility, and transaction-maximum utility are recorded in the AU-list. Thus, only information relevant for HAUIM is stored and the mining efficiency can be improved. MHAI [25] is another HAUIM algorithm that uses a high average-utility itemset (HAI)-list structure. Different from the AU-list, this HAI-list records the maximum item utility of the remaining items, rather than the transaction-maximum utility.

Regardless of the data structure and pruning strategy used, designing a compact and effective upper bound to approximate the final results is universally important for HAUIM algorithms. AUUB was the first upper bound model [5]. It approximates the average-utility of itemsets by summing the maximum utilities of the transactions containing the itemset. The maximum utility of each transaction indicates the highest item utility among the item utilities of all items included in the transaction. Lan et al. improved the AUUB model by removing unpromising items after prefix projection [8]. Lin et al. proposed two tighter upper bounds, one based on the remaining-maximum utility and the other that only considers relevant items [14].

The above upper bounds are either based on the maximal item utilities of the entire transaction or the maximal item utilities of the remaining items with respect to a certain itemset. That is, the object of the basic components consists of the utilities of single items, but not the itemsets.

Wu et al. proposed two upper bounds for HAUIM [23]. In their TUB-HAUPM algorithm, the maximum following utility upper-bound gives the remaining maximal utility of a single item by improving Lin et al.'s upper bound; the top-k revised transaction-maximum utility upper-bound (KRTMUUB) approximates the average-utilities of a k-itemset using a $(k+1)$-itemset. In contrast to KRTMUUB, our proposed algorithm approximates the average-utilities of a k-itemset using only a k-itemset.

4 Proposed algorithm

In this section, we first introduce the proposed generalized maximal utility and GAUUB. Then, we discuss the support pruning strategy and describe the proposed algorithm in detail. Finally, we analyze the complexity of the proposed algorithm.

4.1 Generalized maximal utility and upper bound

Maximal utility is used to overestimate the HAUIs in the original TPAU algorithm [5]. Generally, a summation of the maximal utility of the items in each transaction including the target itemset is used as the initial upper bound. HAUIs are then identified by calculating the actual average-utilities. Regardless of the data structures [10,12] and pruning strategies [9,25] used, the maximal utility is a fundamental component of HAUIM algorithms. In existing methods, maximal utility is only related to the maximal utility of a single item (1-itemset) in each transaction. In this paper, we generalize this concept to k-itemsets.

Definition 4.1 Let $T \in \boldsymbol{D}$ be a transaction with m items. Then, the transaction utility list (TUL) of T is composed of m elements such that each element is a utility value of one item in T. The TUL is defined as follows:

$$TUL(T) = \{u_i \mid for\ any\ 1 \leq i \leq j \leq m, u_i \geq u_j\},$$

where u_i is the utility of a single item in T.

We take T_3 in Table 1 as an example. Here, the items have utilities of $u(A) = 7$, $u(B) = 1$, $u(C) = 3$, and $u(F) = 4$. Thus, $TUL(T_3) = \{7, 4, 3, 1\}$.

Definition 4.2 Let $T \in \boldsymbol{D}$ be a transaction with m items. Then, the generalized maximal utility (GMU) for a k-itemset X such that $X \subseteq T$ is defined as follows:

$$gmu_k(T) = \sum_{j=1}^{k} u_j,$$

where $1 \leq k \leq m$, and u_j is the jth element of $TUL(T)$.

Taking T_3 in Table 1 as an example, the generalized maximal utility of the 2-itemset CF is $gmu_2(T_3) = 7 + 4 = 11$.

From Definition 4.2, we can see that maximal utility [5,25] (or transaction-maximum utility [12,14]) is a special case of generalized maximal utility when $k = 1$. Using generalized maximal utility, we also generalize the average-utility upper bound.

Definition 4.3 Let X be a k-itemset. Then, the generalized average-utility upper bound (GAUUB) of X is defined as follows:

$$gauub(X) = \sum_{X \subseteq T \wedge T \in \boldsymbol{D}} gmu_k(T)/k.$$

Taking CF in Table 1 as an example, there are three transactions, T_1, T_3, and T_4, containing CF, and the TULs of the three transactions are $TUL(T_1) = \{8, 7, 3, 1\}$, $TUL(T_3) = \{7, 4, 3, 1\}$, and $TUL(T_4) = \{9, 4, 4\}$. Thus, according to Definition 4.3, $gauub(CF) = (8 + 7 + 7 + 4 + 9 + 4)/2 = 19.5$. If we use the average-utility upper bound, $auub(CF) = mu(T_1) + mu(T_3) + mu(T_4) = 24$. We can see that GAUUB is tighter than AUUB in this example.

Definition 4.3 shows that the average-utility upper bound [5,12,14,25] is also a special case of GAUUB when $k = 1$.

Theorem 4.1 *Let X be a k-itemset. If $gauub(X) < min\_util$, X is not an HAUI.*

Proof Let T be a transaction with m items, $X \subseteq T$, and $TUL(T) = \{u_1, u_2, \ldots, u_m\}$. We first arrange the items in X in utility-descending order. Let $X = \{x_1, x_2, \ldots, x_k\}$ be the itemset after transformation. Hence, we have, $u(x_1, T) \geq u(x_2, T) \geq \cdots \geq u(x_k, T)$, and $k \leq m$.

$$\begin{aligned} au(X) &= \sum_{X \subseteq T \wedge T \in \boldsymbol{D}} au(X, T) \\ &= \sum_{X \subseteq T \wedge T \in \boldsymbol{D}} u(X, T)/k \\ &= \sum_{X \subseteq T \wedge T \in \boldsymbol{D}} \sum_{p=1}^{k} u(x_p, T)/k \end{aligned}$$

Because $X \subseteq T$, we have $u(x_1, T) \leq u_1, u(x_2, T) \leq u_2, \ldots, u(x_k, T) \leq u_k$. Hence,

$$\begin{aligned} \sum_{X \subseteq T \wedge T \in \boldsymbol{D}} \sum_{p=1}^{k} u(x_p, T)/k &\leq \sum_{X \subseteq T \wedge T \in \boldsymbol{D}} \sum_{p=1}^{k} u_p/k \\ &= \sum_{X \subseteq T \wedge T \in \boldsymbol{D}} gmu_k(T)/k \\ &= gauub(X) \end{aligned}$$

thus, $au(X) \leq gauub(X)$.

Because $gauub(X) < min\_util$, we have $au(X) \leq gauub(X) < min\_util$. Therefore, X is not an HAUI. □

We can see from Theorem 4.1 that GAUUB is an effective metric for overestimating HAUIs.

Theorem 4.2 *Let X be a k-itemset, $gauub(X) \leq auub(X)$*

Proof According to Definitions 4.2 and 4.3, we have

$$\begin{aligned} gauub(X) &= \sum_{X \subseteq T \wedge T \in \boldsymbol{D}} gmu_k(T)/k \\ &= \sum_{X \subseteq T \wedge T \in \boldsymbol{D}} \sum_{j=1}^{k} u(i_j)/k \\ &\leq \sum_{X \subseteq T \wedge T \in \boldsymbol{D}} \sum_{j=1}^{k} max\{u(i_l) | i_l \in T\}/k \\ &= \sum_{X \subseteq T \wedge T \in \boldsymbol{D}} \sum_{j=1}^{k} mu(T)/k \\ &= \sum_{X \subseteq T \wedge T \in \boldsymbol{D}} k \times mu(T)/k \end{aligned}$$

$$= \sum_{X \subseteq T \wedge T \in \boldsymbol{D}} mu(T) = auub(X)$$

□

Theorem 4.2 shows that the proposed GAUUB is tighter than the typical AUUB.

Definition 4.4 Let X be an itemset. We call X a high GAUUB itemset (HGAUUBI) if $gauub(X) \geq min\_util$.

Similarly, an HGAUUBI with k items is called a k-HGAUUBI.

Taking CF in Table 1 as an example, given $min\_util = 16$, since $gauub(CF) = 19.5 > min\_util$, CF is a 2-HGAUUBI.

Theorem 4.3 *Let $SHAUI$ be the set of all HAUIs and $SGHAUI$ be the set of all HGAUUBIs. Then, $SHAUI \subseteq SGHAUI$.*

Proof For $\forall X \in SHAUI$, we have $au(X) \geq min\_util$. According to Theorem 4.1, $gauub(X) \geq au(X)$. Thus, $gauub(X) \geq min\_util$. According to Definition 4.4, $X \in SGHAUI$. Hence, $SHAUI \subseteq SGHAUI$. □

According to Theorem 4.3, for mining HAUIs, we can safely identify HGAUUBIs first and then filter the HAUIs by calculating the actual average-utility of each candidate.

4.2 Support pruning strategy

The pruning strategy is an important component for mining HAUIs efficiently [24,25]. When discovering FIs, the support count of itemset X, denoted $sup(X)$, is defined as the number of transactions in which X occurs as a subset. This quantity is used to improve the mining efficiency of HUI discovery [19]. In the method proposed in this paper, we use the support count to prune the HAUI search space. To realize this strategy, we introduce the concepts of the *maximal utility list* and *critical support count*.

First, we introduce the concept of a *support set*, defined as follows:

$$g(X) = \{T \in \boldsymbol{D} \mid \forall i \in X, i \in T\},$$

where X is an itemset, i is an item, and T is a transaction in database $\boldsymbol{D}$. We can see that a support set associates all the transactions that X appears in as a subset.

Definition 4.5 Let X be an itemset. The itemset maximal utility list (MULS) of X is defined as follows:

$$MULS(X) = \{mu_i \mid for\ any\ i \leq j, mu_i \geq mu_j\},$$

where mu_i is the maximal utility of transaction T_d, and $T_d \in g(X)$.

Definition 4.6 Let $\boldsymbol{D}$ be a transaction database of N transactions. The database maximal utility list (MULD) of $\boldsymbol{D}$ is defined as:

$$MULD(\mathbf{D}) = \{mu_i \mid for\ any\ 1 \leq i \leq j \leq N, mu_i \geq mu_j\},$$

where mu_i is the maximal utility of transaction T_d, and $T_d \in \boldsymbol{D}$.

According to Definitions 4.5 and 4.6, the maximal utility list of itemset X is a set of maximal utilities of the transactions in which X occurs as a subset, and all elements are sorted in descending order of maximal utility. The maximal utility list of database $\boldsymbol{D}$ can be viewed as that of the empty set, that is, $MULS(\emptyset)$.

For example, for the database in Table 1, because $g(CF) = \{T_1, T_3, T_4\}$, $sup(X) = 3$, $MULS(CF) = \{9, 8, 7\}$, and $MULD(\boldsymbol{D}) = \{9, 8, 7, 7, 6\}$.

Definition 4.7 Let $\boldsymbol{D}$ be a transaction database of N transactions and let min_util be the minimum average-utility value. For the database maximal utility list $MULD(\boldsymbol{D}) = \{mu_1, mu_2, \ldots, mu_N\}$, the critical support count is an integer $csup$ such that $\sum_{k=1}^{csup} mu_k \geq min\_util$ and $\sum_{k=1}^{csup-1} mu_k < min\_util$.

According to Definition 4.7, $csup$ is an integer threshold in the database transaction utility list. If lower than $csup$, the cumulative sum of maximal utilities is also lower than min_util; otherwise, the cumulative sum of the maximal utilities is no lower than min_util.

For example, for the database shown in Table 1, if $min\_util = 16$, then $csup = 2$, because for $MULD(\boldsymbol{D})$, $\sum_{k=1}^{2} mu_k \geq min\_util$ and $\sum_{k=1}^{1} mu_k < min\_util$.

Given the critical support count, the following property can be used for pruning the search space of HAUIs.

Theorem 4.4 *Let X be an itemset and $csup$ be the critical support count of transaction database $\boldsymbol{D}$. If $sup(X) < csup$, then X is an LAUI, and any superset of X is also an LAUI.*

Proof Suppose $sup(X) = l$. Because $sup(X) < csup$, we have $l \leq csup - 1 < csup$.

Suppose there are N transactions in $\boldsymbol{D}$, and $MULD(\boldsymbol{D}) = \{mu_1, mu_2, \ldots, mu_N\}$. Similarly, $MULS(X) = \{mu'_1, mu'_2, \ldots, mu'_l\}$. According to Definitions 4.5 and 4.6, $mu'_1 \leq mu_1, mu'_2 \leq mu_2, \ldots, mu'_l \leq mu_l$.

Hence, $au(X) \leq auub(X) = \sum_{X \subseteq T_d} mu(T_d) = \sum_{k=1}^{l} mu'_k = mu'_1 + mu'_2 + \cdots + mu'_l \leq mu_1 + mu_2 + \ldots + mu_l = \sum_{k=1}^{l} mu_k \leq \sum_{k=1}^{csup-1} mu_k$.

According to Definition 4.7, $\sum_{k=1}^{csup-1} mu_k < min\_util$, so we have $au(X) \leq auub(X) < min\_util$. Thus, X is an LAUI.

For all Y such that $X \subset Y$, we have $sup(Y) \leq sup(X)$. Because $sup(X) < csup$, we have $sup(Y) < csup$. Hence, according to the above discussion, Y is also an LAUI.

□

According to Theorem 4.4, given an itemset X, if $sup(X)$ is lower than the critical support count, we can safely prune X and all supersets of X.

4.3 HAUIM-GMU algorithm

The proposed HAUIM-GMU algorithm is described in Algorithm 1. In this algorithm, the extension of an itemset X is based on a set called the tail of X, described as follows.

Let "$\prec$" be the total order on items from $\boldsymbol{I}$. Then, the extensible items of itemset X are called the *tail* of X, denoted as $t(X) = \{i \mid i \in \boldsymbol{I} \wedge \forall j \in X, j \prec i\}$. That is, $t(X)$ is the set of items after X according to the order "$\prec$".

In Algorithm 1, the transaction database is scanned once in Step 1. Then, the pruning strategy proved in Theorem 4.4 is used to eliminate unpromising items with inadequate support counts (Step 2). In Step 3, items are further filtered by deleting those items that are not 1-HGAUUBIs. Next, the transaction utility lists of transactions composed of the remaining

Algorithm 1 HAUIM-GMU

```
Input: Transaction database $\boldsymbol{D}$, minimum average-utility value $min\_util$
Output: All HAUIs
1: Scan database $\boldsymbol{D}$ once;
2: Delete items whose support counts are lower than $csup$;
3: Delete items that are not 1-HGAUUBIs;
4: Calculate the transaction utility list of each transaction;
5: CHAUI-Extension($\emptyset$);
6: for each itemset $is \in SGHAUI$ do
7:     Scan the transactions according to $g(is)$ to calculate $au(is)$;
8:     if $au(is) \geq min\_util$ then
9:         Output $is$;
10:    end if
11: end for
```

items are calculated in Step 4. In Step 5, the procedure CHAUI-Extension (described in Algorithm 2) is called using the empty set as a parameter. The loop from Steps 6-11 checks each itemset in the set of HGAUUBIs (denoted by $SGHAUI$), and outputs the itemsets with average-utility values no lower than min_util. Because of this support set, we can calculate the actual average-utility of each HGAUUBI by scanning the transactions containing it, rather than scanning the whole database.

Algorithm 2 CHAUI-Extension(X)

```
Input: Itemset $X$
Output: HGAUUBIs that include $X$
1: if $t(X) == \emptyset$ then
2:     return ;
3: end if
4: for each item $e \in t(X)$ do
5:     $is = X \cup e$;
6:     if $sup(is) \geq csup$ then
7:         if $gauub(is) \geq min\_util$ then
8:             $is \rightarrow SGHAUI$;
9:         end if
10:        CHAUI-Extension($is$);
11:    end if
12: end for
```

Given an itemset X, Algorithm 2 generates all candidate HAUIs that include X by traversing the search space in a depth-first manner. It terminates if $t(X)$ is empty (Steps 1-3). For each element e in $t(X)$, the main loop enumerates all candidate HAUIs prefixed by X (Steps 4-12). When a new candidate is generated by concatenating an item (Step 5), the critical support count and GAUUB are used sequentially for pruning in Steps 6-7. The qualified itemsets are stored in Step 8, and are used for generating new candidates by calling the CHAUI-Extension procedure recursively (Step 10).

Consider the transaction database in Table 1 and the profit table in Table 2. Suppose that min_util is 16. According to the third column of Table 1, $MULD(\boldsymbol{D}) = \{9, 8, 7, 7, 6\}$, so $csup = 2$. Because $sup(E) = 1$, which is lower than $csup$, it can be pruned from the original database. The GAUUB of each remaining item is listed in Table 3.

Table 3 GAUUB of remaining single items

Item	A	B	C	D	F
guub	22	21	31	15	24

Table 4 Reorganized database

Tid	Transactions	*TUL*
T_1	$(A, 1)(B, 1)(C, 1)(F, 2)$	$\{8, 7, 3, 1\}$
T_2	$(B, 1)$	$\{1\}$
T_3	$(A, 1)(B, 1)(C, 1)(F, 1)$	$\{7, 4, 3, 1\}$
T_4	$(C, 3)(F, 1)$	$\{9, 4\}$
T_5	$(A, 1)(C, 2)$	$\{7, 6\}$

Because $gauub(D) = 15 < min\_util$, item D is also deleted according to Theorem 4.1. The re-organized database is presented in Table 4 with transaction utility lists in the third column.

We take A as an example of candidate extension. In lexicographic order, we have $t(A) = \{B, C, F\}$. We first extend A with item B. Because $sup(AB) = csup$, it passes the support pruning. Because $gauub(AB) = ((8+7)+(7+4))/2 = 13 < min\_util$, AB cannot be an HAUI. Next, AC is considered. It cannot be pruned using the support because $sup(AC) = 3$ is higher than $csup$. Because $gauub(AC) = 19.5 > min\_util$, AC is an HGAUUBI. Similarly, we can verify that AF is not an HGAUUBI.

The above traversal process can be performed in similar fashion for itemsets prefixed by B and C. When all HGAUUBIs have been discovered, the actual average-utility of each candidate is calculated using its support set, rather than by scanning the entire database.

4.4 Complexity analysis

The computational complexity of HAUIM-GMU can be analyzed in terms of the following principal components.

Database scan. During this operation, unpromising single items are deleted and transaction utility lists are calculated. For each transaction, the utility of each item is calculated. Assuming that L is the total number of items in the database, in the worst case, this operation requires $O(N \times L)$ time and space, where N is the total number of transactions. To calculate GAUUB, items are sorted in descending order of utility values. Using quick sort, the time complexity is $O(L \times \log L)$, and the space complexity is $O(L)$. Thus, for the database scan operation, the time complexity is $O(N \times L + L \times \log L)$ and the space complexity is $O(N \times L + L)$.

Generation of HGAUUBIs. HAUIM-GMU generates HGAUUBIs recursively using a pure depth-first search space traversal. Let H be the number of 1- HGAUUBIs. For each path, the maximum depth is upper-bounded by H. As there are at most H paths in the search space, the time complexity should be $O(H^2)$. The space complexity is proportional to the maximum recursion depth, namely $O(H)$.

Identification of HAUIs. After generating all HGAUUBIs, each candidate should be tested to determine whether it is an HAUI or not. Let $| SGHAUI |$ be the number of all HGAUUBIs. In the worst case, all HGAUUBIs are contained in each transaction, so the time complexity is $O(\mid SGHAUI \mid \times N)$. As the database is stored in the database scan stage, the space complexity is $O(\mid SGHAUI \mid)$.

We compare the complexity of HAUIM-GMU with that of three existing algorithms, HAUI-Tree [17], HAUI-Miner [12] and EHAUPM [14]. First, all three algorithms traverse the search space in a depth-first manner, as does the proposed HAUIM-GMU algorithm. Next, we analyze the difference between each of these three algorithms and HAUIM-GMU.

Like HAUIM-GMU, HAUI-Tree is a two-phase algorithm. The candidate results are generated first, and then these candidates are verified to identify the actual results. The main difference between HAUI-Tree and HAUIM-GMU lies in the additional information storage. For each candidate, HAUI-Tree stores the position of the last item in the transactions containing it, and so the space complexity is proportional to the number of HAUUBIs. Assuming that L is the total number of items in the database, in the worst case, the space complexity of this operation is $O(2^L - 1)$. For HAUIM-GMU, if N is the total number of transactions, the space for storing the transaction utility list is $L \times N$, and the space for storing the database maximal utility list is N. Thus, the space complexity of this operation is $O(L \times N + N)$. Thus, we can see that HAUIM-GMU is superior to HAUI-Tree for the storage of additional information.

We further analyze and compare the space complexity of HAUI-Miner and EHAUPM with HAUIM-GMU. For HAUI-Miner, three fields are stored in one node, so the storage space in the worst case is $3 \times (2^L - 1)$. For EHAUPM, four fields are stored in one node, leading to a storage space of $4 \times (2^L - 1)$; furthermore, the estimated average-utility co-occurrence matrix (EAUCM) is used to store AUUB values of 2-itemsets. Thus, in the worst case, the storage space of EAUCM is $(L \times L)/2$. Thus, the storage space for additional information of EHAUPM is $4 \times (2^L - 1) + (L \times L)/2$. According to the above discussion, regardless of whether an AU-list or MAU-list is used, in the worst case, the space complexity of HAUI-Miner and EHAUPM is basically $O(2^L - 1)$. According to the above discussion, for HAUIM-GMU, the space complexity of storing additional information is $O(L \times N + N)$. Thus, HAUIM-GMU is superior to these two algorithms for the storage of additional information. However, HAUIM-GMU needs to test the GAUUBIs, and the complexity of this operation is $O(2^L - 1)$ in the worst case. For HAUI-Miner and EHAUPM, no candidates are generated. In general, considering both the storage of additional information and the testing of candidates, the complexity of HAUIM-GMU is comparable to that of HAUI-Miner and EHAUPM. The actual performance of HAUIM-GMU depends on the effect of the GAUUB model and the support pruning strategy.

5 Experimental results

We evaluated the performance of our algorithm and compared it with that of three HAUIM algorithms, HAUI-Tree [17], HAUI-Miner [12], and EHAUPM [14]. The source code for HAUI-Miner and EHAUPM was downloaded from the SPMF data mining library [3].

5.1 Experimental environment and datasets

The experiments were performed on a computer with a 4-Core 3.40 GHz CPU and 8 GB memory running 64-bit Microsoft Windows 10. Our programs were written in Java. Six datasets were used to evaluate the performance of the algorithms. The characteristics of the datasets are presented in Table 5.

Table 5 Characteristics of the datasets

Datasets	Avg.Trans.Len.	#Items	#Trans
T26I80D4K	26	80	4000
T25I200D10K	25	200	10,000
Chess	37	76	3196
Connect	43	130	67,557
Retail	10.3	16,470	88,162
Mushroom	23	119	8124

T26I80D4K and T25I200D10K are synthetic datasets produced using the data generator provided by SPMF [3]. The parameters used are T, I, and D, which represent the average item length per transaction, total number of different items, and total number of transactions, respectively. The other four datasets were downloaded from SPMF [3]. The Chess dataset contains game steps from several real-life chess matches. The Connect dataset is also derived from game steps. The Retail dataset contains anonymized retail-market basket data from a Belgian retail store. The Mushroom dataset contains various species of mushrooms and characteristics such as shape, odor, and habitat.

5.2 Performance analysis on synthetic datasets

In this section, we evaluate the performance of the four algorithms on the synthetic datasets T26I80D4K and T25I200D10K. The runtime and memory were tested under various minimum average-utility thresholds, and the comparison results are shown in Fig. 1. The peak memory usage was measured using the standard Java API.

For the comparison results on T26I80D4K shown in Fig. 1a, b, we can see that HAUIM-GMU achieves a considerable performance improvement compared with the other three HAUI mining algorithms. In terms of efficiency, HAUIM-GMU is consistently an order of magnitude faster than both HAUI-Tree and HAUI-Miner, and twice as fast as EHAUPM. Although the superior efficiency of HAUIM-GMU over EHAUPM is not as obvious as for the other two algorithms, the gap between them increases as the minimum average-utility threshold decreases. In terms of memory consumption, HAUIM-GMU consumes, on average, less memory than the other three algorithms. The memory consumption of HAUI-Tree and HAUI-Miner fluctuates. For example, when the threshold is 1.00%, HAUI-Tree has the lowest memory requirements and HAUI-Miner has the highest. However, when the threshold is 0.98%, the situation is reversed, with HAUI-Tree consuming the most memory and HAUI-Miner using the least. The main difference between these two algorithms is that HAUI-Tree stores the position of the last item of each candidate itemset, whereas HAUI-Miner uses AU-list and the pruning strategies. For the two thresholds of 1.00% and 0.98%, the AU-list and pruning strategies perform worst and best alternately, and the relative positions are correspondingly worst and best. On average, EHAUPM performs the worst with respect to memory usage for the T26I80D4K dataset. This is because the MAU-list and EAUCM store more information for pruning. However, these structures do not reduce the search space significantly on T26I80D4K.

Comparing the results for T25I200D10K shown in Fig. 1c, d, HAUIM-GMU is consistently faster than the other three algorithms. Furthermore, HAUIM-GMU remains constant at around 10 s, rather than increasing as the minimum threshold varies between 0.20 and 0.12%.

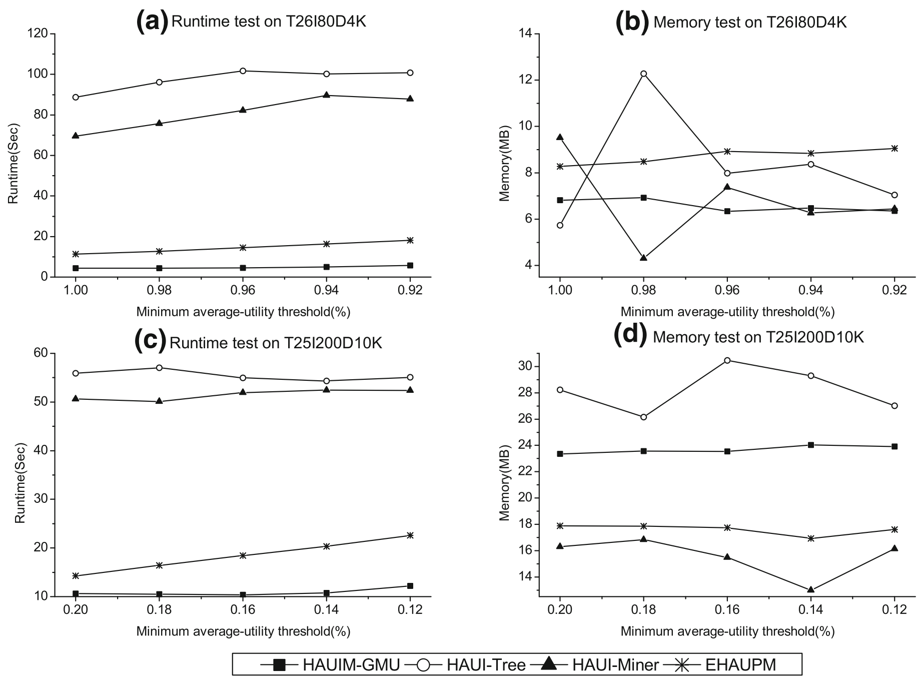


Fig. 1 Performance evaluation on synthetic datasets

This indicates that the proposed GAUUB model is effective at steadily filtering candidate HAUIs. In terms of memory usage, HAUI-Miner performs the best, followed by EHAUPM, HAUIM-GMU and HAUI-Tree in that order. This is because, when the minimum average-utility threshold is between 0.20 and 0.12%, most HAUIs contain fewer than two items. For example, when the threshold is 0.12%, there are 3624 HAUIs: 4.8% of them are 1-HAUIs and 95.2% are 2-HAUIs. In these circumstances, neither the MAU-list used in EHAUPM [14] nor the GAUUB in HAUIM-GMU are as effective as the AU-list of HAUI-Miner [12] at reducing the memory usage.

From the above discussion, we can see that HAUIM-GMU outperforms HAUI-Tree, HAUI-Miner, and EHAUPM with regard to efficiency, and requires considerably less memory when applied to synthetic datasets.

5.3 Performance analysis on real datasets

In this section, we evaluate the performance of the four algorithms on four real datasets. The comparison results in Fig. 2 show that HAUIM-GMU produces the best performance with all datasets except Retail.

Comparing the results for the Chess dataset, shown in Fig. 2a, b, we can see that the runtime increases as the minimum threshold decreases. On average, HAUIM-GMU is 5.57 times faster than EHAUPM, one order of magnitude faster than HAUI-Miner, and two orders of magnitude faster than HAUI-Tree. HAUIM-GMU consistently consumes less memory than the other three algorithms. HAUI-Miner uses less memory than EHAUPM when the minimum threshold is less than 4.60%. Similar to the results shown in Fig. 1b, the memory

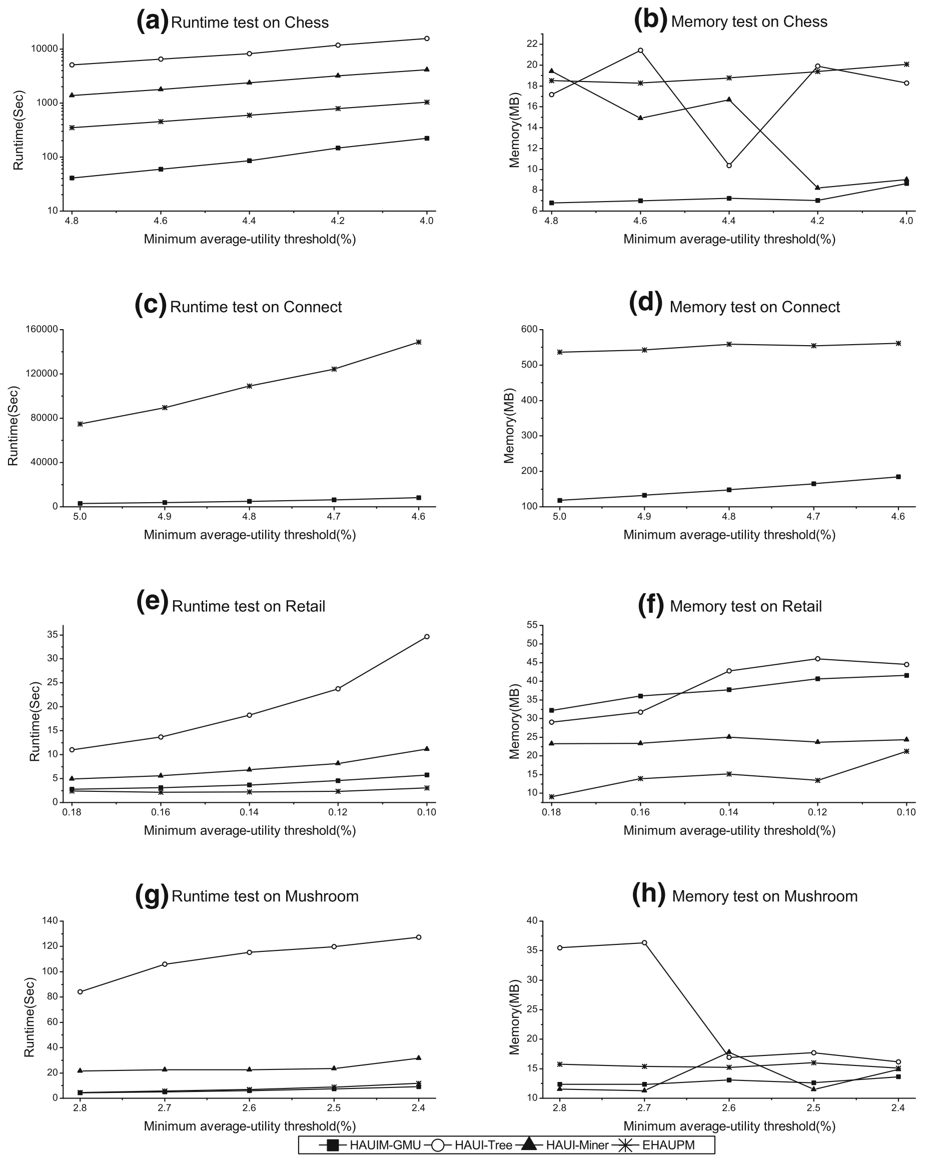


Fig. 2 Performance evaluation on real datasets

usage of HAUI-Tree and HAUI-Miner fluctuate, which indicates that the tree sizes and AU-list structures vary with the minimum threshold. The improved memory performance of HAUIM-GMU reflects the tighter upper bound of the GAUUB model, which reduces the number of approximate results, and demonstrates that the critical support count is effective at filtering out unpromising items.

Figure 2c, d compares the results with the Connect dataset. The superiority of HAUIM-GMU is more obvious in this case. For all thresholds in Fig. 2c, d, both HAUI-Tree and HAUI-Miner could not return any results after 50 h; thus, the results for these two algorithms are not plotted. In terms of efficiency, HAUIM-GMU is always one order of magnitude faster than EHAUPM. For example, when the minimum high average-utility threshold is set to 4.8%, HAUIM-GMU and EHAUPM terminate after 4950 s and 108,991 s, respectively. In terms of memory storage, both algorithms exhibit a steady increase as the minimum high average-utility threshold decreases. This is reasonable because, when the minimum high average-utility threshold is lower, more itemsets are HAUIs. Thus, the search space is increased, as is the memory usage. On average, HAUIM-GMU requires 2.76 times less memory than EHAUPM. This indicates that the great efficiency improvement is not the result of sacrificing more storage space.

Of the four real datasets, Retail is the only one for which the proposed HAUIM-GMU did not outperform the other algorithms. As the results in Fig. 2e, f show, HAUIM-GMU is slower than EHAUPM and consumes more memory than EHAUPM and HAUI-Miner. By analyzing the characteristics of the Retail dataset, the following was identified as causing the weaker performance of HAUIM-GMU. Table 5 indicates that Retail is a very sparse dataset, with the shortest average transaction length among all six datasets considered in this study. In this dataset, most HAUIs are very short. For example, when the minimum average-utility threshold is 0.16%, all HAUIs contain three or fewer items. Specifically, 60% of HAUIs are 1-HAUIs, 27% of results are 2-HAUIs, and 13% are 3-HAUIs. In this case, the GAUUB model can only omit a limited number of items, leading to low efficiency and more storage space.

Finally, the results for the Mushroom dataset are shown in Fig. 2g, h. HAUIM-GMU performs best in terms of both efficiency and storage usage, but the superiority is not obvious. In terms of runtime, HAUIM-GMU is slightly faster than EHAUPM. Regarding memory usage, HAUIM-GMU consumes slightly less memory than HAUI-Miner. These results are because Mushroom contains many long itemsets with most items occurring together. Thus, the support count pruning is not effective for most items with high supports, and so neither the mining efficiency nor memory usage are significantly improved by the critical support count.

In summary, the proposed HAUIM-GMU algorithm outperforms the related HAUI mining algorithms on all real datasets considered here, except Retail. Generally, the search space can be considerably reduced by adopting tighter upper-bounds, and the mining speed can be improved using the support pruning strategy.

5.4 Evaluation of pruning strategy

To evaluate the benefit of the support pruning strategy, we compared the performance of HAUIM-GMU with and without this technique. The implementation without a support pruning strategy is referred to as HAUIM-GMU-B. We conducted experiments on the T25I200D10K synthetic dataset and the Connect and Retail real datasets. T25I200D10K was chosen because it is larger than its synthetic counterpart T26I80D4K. The two real datasets were selected because, of the two that HAUIM-GMU produced very good performance, Connect is larger than Chess. Furthermore, Retail was chosen because HAUIM-GMU did not outperform the other three algorithms with this dataset. The comparison results are shown in Fig. 3.

As shown in Fig. 3a, b, on the T25I200D10K dataset, the support pruning strategy accelerates the mining efficiency by one order of magnitude. Furthermore, memory consumption is also reduced.

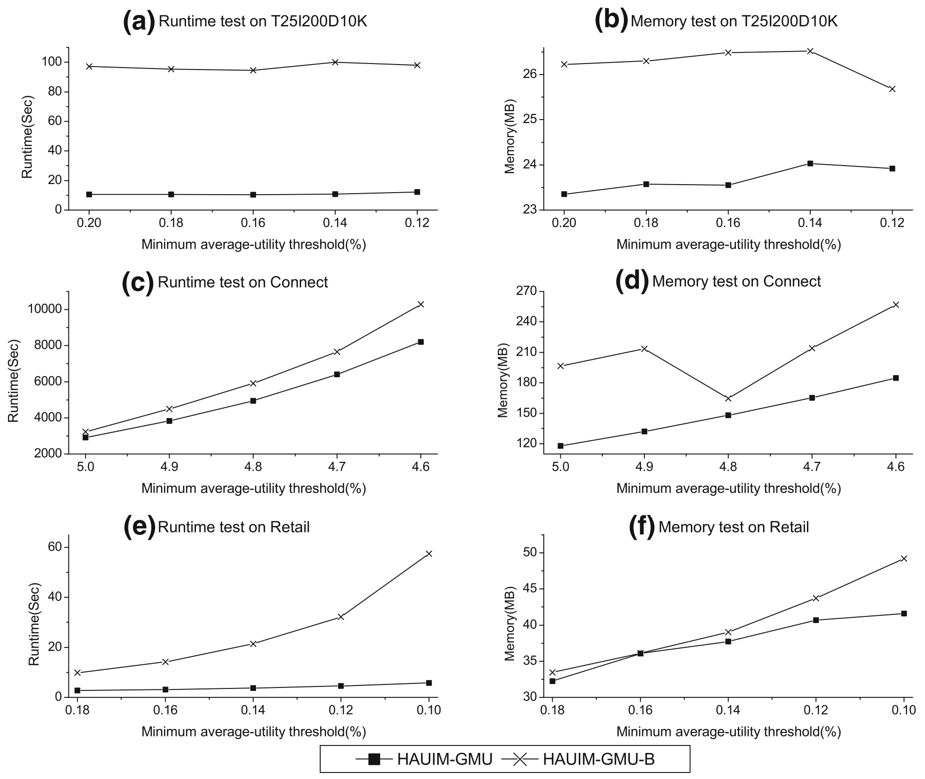


Fig. 3 Performance evaluation with and without the support pruning strategy

In Fig. 3c, d, although the difference between HAUIM-GMU and HAUIM-GMU-B is not as significant as for T25I200D10K, both runtime and memory consumption are better when the support pruning strategy is applied to the Connect dataset.

Although HAUIM-GMU did not outperform the other three algorithms with the Retail dataset, the results shown in Fig. 3e, f demonstrate that the support pruning strategy can still improve performance and reduce storage space. For example, when the minimum average-utility threshold is set to 0.10%, using the support pruning strategy can produce an efficiency improvement of one order of magnitude.

According to the results shown in Fig. 3, we can see that the support pruning strategy is more suitable for sparse datasets such as T25I200D10K and Retail. The main reason is that the support of items in this kind of dataset is usually low. Thus, the critical support can filter out lots of unpromising items recursively. However, for dense datasets like Connect, most items' supports are high, so few items can be pruned using the critical support. Based on these results, we conclude that the support pruning strategy can improve both runtime efficiency and memory consumption.

5.5 Effectiveness of GAUUB

To verify the superiority of GAUUB over AUUB, we compared the number of HGAUUBIs generated by GAUUB with the number of HAUUBIs generated by AUUB. The results are

Table 6 Comparison of the number of HGAUUBIs to HAUUBIs for the T26I80D4K dataset

Minimum average-utility threshold (%)	NG	NH	RR (%)
1	3758	69,349	94.58
0.98	4625	75,007	93.83
0.96	6405	79,236	91.92
0.94	9689	81,983	88.18
0.92	15,071	83,626	81.98
Average			90.10

Table 7 Comparison of the number of HGAUUBIs to HAUUBIs for the T25I200D10K dataset

Minimum average-utility threshold (%)	NG	NH	RR (%)
0.2	19,900	19,900	0
0.18	19,900	19,900	0
0.16	19,900	19,900	0
0.14	19,900	19,900	0
0.12	19,900	19,900	0
Average			0

Table 8 Comparison of the number of HGAUUBIs to HAUUBIs for the Chess dataset

Minimum average-utility threshold (%)	NG	NH	RR (%)
4.80	46,352	94,245	50.82
4.60	67,333	134,434	49.91
4.40	97,745	191,932	49.07
4.20	170,348	325,126	47.61
4.00	253,137	474,712	46.68
Average			48.82

presented in Tables 6, 7, 8, 9, 10 and 11. In these tables, the values in the final column represent the reduced ratio (RR), which is computed as:

$$RR = (1 - NG/NH) \times 100\%,$$

where *NG* and *NH* are the numbers of HGAUUBIs and HAUUBIs, respectively.

Tables 6-11 indicate that compared with a typical AUUB model, the proposed GAUUB model reduces the number of candidates, except on the T25I200D10K dataset. As we explained in Sect. 5.2, most HAUIs have fewer than two items when the minimum threshold is between 0.20 and 0.12% on T25I200D10K. Thus, GAUUB cannot filter more candidates than AUUB. Furthermore, the filter effect of GAUUB on the Retail dataset is not obvious, with an average below 20%. For the Chess and Connect datasets, the GAUUB model halves the number of candidates, on average, compared with an AUUB model. The effect of the GAUUB model is obvious on T26I80D4K and Mushroom. For these two datasets, the GAUUB model reduces the number of candidates obtained by the AUUB model by an average of 90.10% and 87.45%, respectively.

Table 9 Comparison of the number of HGAUUBIs to HAUUBIs for the Connect dataset

Minimum average-utility threshold (%)	NG	NH	RR (%)
5.00	72,755	163,161	55.41
4.90	96,631	209,930	53.97
4.80	127,739	269,105	52.53
4.70	169,366	345,867	51.03
4.60	223,881	442,696	49.43
Average			52.47

Table 10 Comparison of the number of HGAUUBIs to HAUUBIs for the Retail dataset

Minimum average-utility threshold (%)	NG	NH	RR (%)
0.18	225	261	13.79
0.16	282	327	13.76
0.14	364	442	17.65
0.12	479	573	16.40
0.10	669	840	20.36
Average			16.39

Table 11 Comparison of the number of HGAUUBIs to HAUUBIs for the Mushroom dataset

Minimum average-utility threshold (%)	NG	NH	RR (%)
2.80	4120	37,362	88.97
2.70	4928	53,215	90.74
2.60	6128	54,280	88.71
2.50	7780	55,661	86.02
2.40	10,043	58,427	82.81
Average			87.45

According to the above analysis, we can see that the proposed GAUUB model is an effective means of generating fewer candidates than the typical AUUB model, as long as the resulting HAUIs are not very short in length.

6 Conclusions and future work

In this paper, we have extended the concept of maximal utility and average-utility upper bound from an item to an itemset, and proposed a new HAUIM algorithm called HAUIM-GMU. We discussed the rationality of the new models and exploited the concept of support for pruning. We compared the performance of the proposed algorithm against three state-of-the-art algorithms, and the experimental results show that HAUIM-GMU is efficient and requires less memory.

Although the proposed GAUUB model is tighter than the dominant AUUB model, and is an effective method for filtering out unpromising itemsets, the HAUIM-GMU algorithm still follows the two-phase routine that requires one more stage of candidate verification.

In future work, we will attempt to incorporate a novel data structure to avoid candidate generation. Furthermore, extending the HAUIM-GMU algorithm to big data applications will be investigated in future studies.

Acknowledgements The authors would like to thank the anonymous reviewers for their valuable comments and suggestions, which helped to improve the quality of this paper. This work was partially supported by the National Natural Science Foundation of China (61977001) and the Great Wall Scholar Program (CIT & TCD20190305).

Declarations

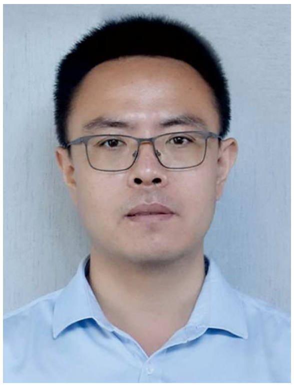
Conflict of interest The authors declare that they have no conflict of interest.

References

1. Agrawal R, Srikant R (1994) Fast algorithms for mining association rules in large databases. In: Proceedings 20th international conference on very large data bases. Morgan Kaufmann, Santiago de Chile, pp 487–499
2. Deng Z-H (2018) An efficient structure for fast mining high utility itemsets. Appl Intell 48(9):3161–3177
3. Fournier-Viger P, Lin CW, Gomariz A, Gueniche T, Soltani A, Deng Z, Lam HT (2016) The SPMF open-source data mining library version 2. In: Proceedings of the 19th European conference on machine learning and knowledge discovery in databases, Riva del Garda, Italy (September 2016) Lecture notes in computer science, vol 9853. Springer, Cham, pp 36–40
4. Han J, Pei J, Yin Y, Mao R (2004) Mining frequent patterns without candidate generation: a frequent-pattern tree approach. Data Min Knowl Discov 8(1):53–87
5. Hong T-P, Lee C-H, Wang S-L (2009) Mining high average-utility itemsets. In: Proceedings of the 2009 IEEE international conference on systems, man, and cybernetics. IEEE, San Antonio, pp 2526–2530
6. Jaysawal BP, Huang J-W (2019) DMHUPS: discovering multiple high utility patterns simultaneously. Knowl Inf Syst 59(2):337–359
7. Kim D, Yun U (2017) Efficient algorithm for mining high average-utility itemsets in incremental transaction databases. Appl Intell 47(1):114–131
8. Lan G-C, Hong T-P, Tseng VS (2012) Efficiently mining high average-utility itemsets with an improved upper-bound strategy. Int J Inf Tech Decis 11(5):1009–1030
9. Lan G-C, Hong T-P, Tseng VS (2012) A projection-based approach for discovering high average-utility itemsets. J Inform Sci Eng 28:193–209
10. Lin C-W, Hong T-P, Lu W-H (2010) Efficiently mining high average utility itemsets with a tree structure. In: Proceedings of the second international conference on intelligent information and database systems, Hue City, Vietnam (March 2010). Lecture notes in computer science, vol 5990. Springer, Berlin, pp 131–139
11. Lin J C-W, Li T, Fournier-Viger P, Hong T-P, Su J-H (2016) Efficient mining of high average-utility itemsets with multiple minimum thresholds. In: Proceedings of the industrial conference on data mining, New York, NY, USA (July 2016). Lecture notes in computer science, vol 9728. Springer, Cham, pp 14–28
12. Lin JC-W, Li T, Fournier-Viger P, Hong T-P, Zhan J, Voznak M (2016) An efficient algorithm to mine high average-utility itemsets. Adv Eng Inform 30(2):233–243
13. Lin JC-W, Ren S, Fournier-Viger P (2018) MEMU: more efficient algorithm to mine high average-utility patterns with multiple minimum average-utility thresholds. IEEE Access 6:7593–7609
14. Lin JC-W, Ren S, Fournier-Viger P, Hong T-P (2017) EHAUPM: efficient high average-utility pattern mining with tighter upper bounds. IEEE Access 5:12927–12940
15. Lin JC-W, Shao Y, Fournier-Viger P, Djenouri Y, Guo X (2018) Maintenance algorithm for high average-utility itemsets with transaction deletion. Appl Intell 48(10):3691–3706
16. Liu Y, Liao W-K, Choudhary AN (2005) A two-phase algorithm for fast discovery of high utility itemsets. In: Proceedings of the 9th Pacific-Asia conference on advances in knowledge discovery and data mining, Hanoi, Vietnam (May 2005). Lecture notes in computer science, vol 3518. Springer, Berlin, pp 689–695
17. Lu T, Vo B, Nguyen H, Hong T-P (2015) A new method for mining high average utility itemsets. In: Proceedings of the 13th IFIP international conference on computer information systems and industrial management. Springer, Ho Chi Minh City, pp 33–42

18. Sethi KK, Ramesh D, Sreenu M (2019) Parallel high average-utility itemset mining using better search space division approach. In: Proceedings of the international conference on distributed computing and internet technology, Bhubaneswar, India (January 2019). Lecture notes in computer science, vol 11319. Springer, Cham, pp 108–124
19. Song W, Liu Y, Li JH (2014) Mining high utility itemsets by dynamically pruning the tree structure. Appl Intell 40(1):29–43
20. Song W, Liu Y, Li JH (2014) BAHUI: fast and memory efficient mining of high utility itemsets based on bitmap. Int J Data Warehous 10(1):1–15
21. Song W, Yang BR, Xu ZY (2008) Index-BitTableFI: an improved algorithm for mining frequent itemsets. Knowl-Based Syst 21(6):507–513
22. Song W, Zhang Z, Li JH (2016) A high utility itemset mining algorithm based on subsume index. Knowl Inf Syst 49(1):315–340
23. Wu JM-T, Lin JC-W, Pirouz M, Fournier-Viger P (2018) TUB-HAUPM: tighter upper bound for mining high average-utility patterns. IEEE Access 6:18655–18669
24. Wu R, He Z (2018) Top-k high average-utility itemsets mining with effective pruning strategies. Appl Intell 48(10):3429–3445
25. Yun U, Kim D (2017) Mining of high average-utility itemsets using novel list structure and pruning strategy. Future Gen Comp Syst 68:346–360
26. Yun U, Kim D, Ryang H, Lee G, Lee K-M (2016) Mining recent high average utility patterns based on sliding window from stream data. J Intell Fuzzy Syst 30(6):3605–3617
27. 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

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



Wei Song received his Ph.D. degree in Computer Science from University of Science and Technology Beijing, Beijing, China, in 2008. He is a professor in School of Information Science and Technology at North China University of Technology. His research interests are in the areas of data mining and knowledge discovery. He has published more than 40 research papers in refereed journals and international conferences.

Lu Liu received her B.E. degree in Software Engineering from North University of China, Taiyuan, China, in 2017, and M.E. degree in Computer Science and Technology from the North China University of Technology, Beijing, China, in 2020. Her research interests include pattern mining and heuristic search.

Chaomin Huang received his B.E. degree from Harbin Engineering University, Harbin, China, in 2014, and M.E. degree from North China University of Technology, Beijing, China, in 2019. His research interests are in the areas of data mining and knowledge discovery.