Applied Intelligence (2023) 53:6099–6118
https://doi.org/10.1007/s10489-022-03722-x

Efficient mining high average-utility itemsets with effective pruning strategies and novel list structure

Gufeng Li[1,2] · Tao Shang[1,2] · Yinling Zhang[1,2]

Accepted: 5 May 2022 / Published online: 6 July 2022

Abstract
High utility itemset mining can mine all itemsets that meet the minimum utility threshold set by the decision maker, thus becomes a popular and prominent data-mining technique. High average utility itemset mining(HAUIM) can determine the desired pattern by considering the utility and length of itemset, which is a fairer alternative measurement. Recently, several algorithms use various utility-list structures and upper bounds to improve the pruning methods. However, these algorithms generated too many unpromising candidates, since they overestimated the average-utility of itemsets too much. This paper designs a novel utility list structure and presents an efficient upper-bound model for improving the performance of HAUIM methods. This list structure captures all key feature informations to estimate the tighter upper-bound of itemset average-utility, it also can be used to calculate the actual itemset average-utility. For avoiding the processing of unpromising candidates, this pruning strategy utilizes the tighter upper-bound. Thus it reduces the number of join operations greatly compared with the state-of-art HAUIM methods. Various experiments were performed by several benchmark datasets to measure the performances of proposed algorithm. The experimental results show that the proposed algorithm has runtime, memory consumption, number of join operations, and scalability performances superior to those of existing algorithms.

Keywords High average-utility pattern · Utility List · Efficient upper bounds · Data mining

1 Introduction

With the increasing interest in data analysis, data mining technique has become a popular technique, since it can mine important, interesting and potential useful information in large databases. Association rule mining can discover the meaningful relationships between different items from large-size and cluttered databases, which is a major research topic in the field of data mining.

And association rules can be generated by calculating the support and confidence between frequent itemsets, so the problem of association rules mining can be attributed to frequent itemsets mining.

In a given transaction database, association rule mining [7, 11, 13, 47] finds the itemsets according to the support of each item, which can be considered as the real-life elements, for example, the commodities and error codes in the market and operation system, respectively. The meaningful itemsets are useful in decision making processes and helpful to classification, clustering and other data mining techniques. Frequent itemset mining can discover the itemsets that are frequently purchased by customers in a transaction database, which is one of its basic techniques. Apriori [1] and FP-Growth [12] are typical of its algorithms. Meanwhile, various extended applications based on frequent itemset mining are presented to solve the different application problems of various databases, such as rare/correlated itemsets mining [3, 34], closed pattern mining [4, 39], and frequent itemset mining in a stream [30, 43].

Tao Shang and Yinling Zhang contributed equally to this work.

✉ Gufeng Li
gufengli@stu.xidian.edu.cn

Tao Shang
tshang@xidian.edu.cn

Yinling Zhang
yinlingzhang@stu.xidian.edu.cn

[1] State Key Laboratory of Integrated Service Network, Xidian University, Xi'an, Shaanxi, China

[2] Collaborative Innovation Center of Information Sensing and Understanding, Xidian University, Xi'an, Shaanxi, China

However, for the meaningful evaluation of the itemset, frequent itemset mining has several limitation. It neglects that the item has unique profit and quantity parameters in the real-life besides frequency parameter. It means that the measurement of the importance of every item is not fully considered. In frequent itemset mining, once an item appears in a transaction, then it has the same importance as the other items in the transaction. The different profits and quantities of items will influence the importance of items especially in market databases with information of product profits and quantities. For example, the smartphone with high profit and low quantity, and the headphone with low profit and high quantity should have the approximate importance value.

For overcoming the limitation of frequent itemset mining, HUIM [8, 9, 27, 31, 32, 42, 46] was proposed to further improve the evaluation of important information. It incorporates internal and external utility of each item to evaluate the itemset value and perform its mining process. However, because the utility of item is calculated by the internal and external utility, the DC property of frequent itemset mining can't be applied directly.

For solving this problem, the transaction-weighted utility(TWU) model was proposed, which determines the upper-bound utility for an itemset. To solve this problem, [24] proposed the transaction-weighted utility(TWU) model, which defines an upper-bound on the utility of itemsets.

It is a new downward closure property, which can help HUIM algorithms reduce the search space and consume less runtime to mine all high utility itemsets(HUIs). This property and additional improvements were applied in several algorithm [10, 24] to speed up the performance of HUIM algorithms.

Even though the problems of the frequent itemset mining were solved by HUIM, some new drawbacks arose in the meantime. HUIM perform the process of the mining by calculate the sum of each item utility. Therefore, the itemsets with longer length are more likely to be mined because they have high probability of higher utilities. Due to the limitation of user analysis, the itemsets with long length are difficult to reveal meaningful information. High average utility itemset mining [14, 17, 26] is a more equitable approach proposed recently, which can solve the problem and fully evaluate the importance of the itemset. The utility of an itemset divided by its length(the number of items it contains) is equal to the average-utiltiy of the itemset. Therefore, the HUIs with long length cannot have high importance, the low utility itemset with short length and high unit utility will be more likely to be discovered.

And in the HAUIM, the average-utility upper-bound(AUUB) model was the revised DC property, which is equivalent to the TWU model in the HUIM. It ensure the correctness of depth-first iterative mining and the completeness of the search space.

In this paper, we propose a new HAUIM algorithm, named efficient mining high average-utility itemsets(EMAUI), for mining HAUIs from a large-size transaction database. Compared with the existing HAUIM algorithms such as HAUI-Miner, EHAUPM, and MHAI, this algorithm is more efficient. The HAUI-Miner [19] was the first one-phase HAUIM algorithm, it proposed average-utility list to store the utility of items and applied the pruning strategy to determine the HAUIs. Due to the loose upper bound offered by AUUB model overestimating, HAUI-Miner generates so many candidates and performs too many costly join operation. The MHAI [45] and EHAUPM [20] algorithms improved those issues. EHAUPM proposed LUB and RTUB method to reduce the upper bound of average-utility. At the same time, combined with some pruning strategies, it removed the unpromising candidate itemsets significantly. MHAI proposed High Average-utility Itemset list to store the *mu* and *mn* parameters. Relying the corresponding pruning strategies, it improved the efficiency of HAUIM process.

Although they have proposed new structures and upper bounds, they both still overestimate the upper bound of average-utility and failed to effectively avoid the occurrence of irrelevant candidate itemsets. If the search space of itemset can be further decreased, then it is worth storing more information in the list structure. On the basis of this idea, we propose a novel list structure to obtain more information about the characteristics of items in each transaction to advance the previous algorithms.

In additions, for speeding up the mining process, the tighter upper bound and corresponding effective pruning strategies are used in the proposed algorithm, which can further avoid mining unnecessary itemsets and reduce the join counts.

In order to evaluate the performance of the proposed algorithm, six standard datasets, which are used in many data mining researches [5], were used for running time, memory usage, number of join operation, and scalability experiments. The major contributions of this paper are summarized as follows:

1. A new list structure named KFAU-List (Key feature average-utility list) is designed. KFAU-List obtain the key information about the features of each item in each transaction, and then helps the proposed method to calculate the tighter upper bound and avoid generation of unnecessary candidates. This list structure consumes a little more memory for capturing more feature information, but it reduces the join count greatly, thus the maximum memory consumption is not much different from other algorithms in most cases.
2. The proposed algorithm proposes an effective strategy to prune the search space of itemsets. It introduces the

relative maximum average-utility upper-bound(RMUB) to estimate the tighter upper bound to determine whether the itemset is explored. It can avoid the generation of unnecessary candidates during the mining process.
3. For further avoiding the occurrence of irrelevant candidate itemsets, the Early Abandoning(EA) strategy is adopted for abandoning KFAU-list construction when the itemset is judged to be unpromising.
4. Various experiments were performed by several benchmark datasets to measure the performance of proposed algorithm. Experiments with six standard datasets, comparing the runtime, memory, join counts, RMUB and EA efficiency, and scalability performance of EMAUI algorithm with those of state-of-art HAUIM algorithms.

The rest of this paper contains the following parts. Section 2 introduces the background knowledge of related work. The basic concepts and latest algorithms of HUIM and HAUIM are briefly described. In Section 3, the designed KFAU-list structure, the tighter upper-bound along with the adopted pruning strategies, and the proposed HAUIM algorithm are described in detail. In Section 4, massive experiments are performed to measure the performance of our algorithm. Finally, conclusions and the direction of future work are given in Section 5

2 Related work

2.1 High utility itemset mining

For efficient performing the HUIM processes, the downward closure property of the itemset is an important problem to be solved. The Two-Phase algorithm [25] is the first algorithm to propose the TWU upper-bound. According to the TWU model, the algorithm can exclude items that have no hope of becoming HUIs before performing mining process. Meanwhile, Two-Phase algorithm requires repeated scanning of the dataset before determining that the itemset is HUI. And due to the loose upper-bound, the numerous candidates are generated that need to be considered to discover the HUIs. Similar to FP-Growth algorithm, IHUP [2], UP-Growth [37], and UP-Growth+ [38] store data based tree structure to avoid repeated database scanning. According to their tree data structure, these algorithms use recursive pattern growth approach to discover the candidate itemsets. Nevertheless, these algorithms still need additional database scan to finally determine the actual utility of each candidate itemset. In order to mine HUIs without the generation of candidate itemsets, the HUI-Miner [24] method, which is the first One-Phase algorithm, was proposed. This algorithm proposes list structure and the enumeration tree, which are utilized to store the actual utility and recursively perform the high utility itemset mining, respectively. After that, there are a series of algorithms based on list structure, such as FHM [6], HUP-Miner [24], EFIM [48] and HMiner [16]. These algorithms improve the performance of list structure HUIM algorithm according to their advanced list structure and the revised pruning strategies.

2.2 High average-utility itemset mining

HAUIM can determine the desired pattern by considering the utility and length of itemset and evaluate the importance of itemsets more fairly in HAUI mining process. TPAU [14], which is the first HAUIM algorithm, has a similar framework to Apriori and performed its mining process in two-phase. For maintaining the downward closure property similar to TWU, TPAU firstly proposed AUUB model to overestimate the average utility of itemset. According to the AUUB, the unpromising itemset is no need to be extended. However, due to the Apriori-like framework, this algorithm also has to scan dataset repeatedly. Afterwards, there are many other algorithms that have been proposed to mine the average-utility itemsets. The HAUP-growth [18] algorithm store the actual average-utility of each itemset in a tree data structure. And according to the array stored in each node, it utilizes the recursive pattern growth approach to mine all HAUIs without repeated data scanning. The PBAU [17] algorithm only stores the specific itemset identify in the indexing structure. In addition to the projection-based technique, it speeds up the performance of mining process. HAUI-tree [26] uses the revised tree structure to improve the calculation, which is another HAUIM method. Recently, several algorithms have used various utility-list structures and upper bounds to improve the HAUIM methods. HAUI-Miner [19] algorithm proposes average-utility list to store the utility of items and applied the pruning strategy based on AUUB model to determine the HAUIs. EHAUPM [20] algorithm proposes LUB and RTUB method to reduce the upper bound of average-utility. In addition to the corresponding pruning strategies, the algorithm removes the unpromising candidate itemsets significantly. Meanwhile, MHAI [45] offers a novel list structure HAI-list to capture more information. Combined with the tighter pruning strategies, it can estimate the average-utility of the super itemsets before the branch of itemset is mined.

Recently, a few more research works such as dHAUIM [35], VMHAUI [36], and HAUL-Growth [44] also improve performance recently according to the tighter average upper-bound, but they are all tree-based HAUI algorithms. They need to generate the conditional trees or projected databases, which will consume a large amount of time. To solve the problems of tree-based HAUIM algorithms, these paper still uses the utility list structure to mine the HAUIs.

FHAUM [21] proposes two versions: D-FHAUM utilizes depth-first search, while B-FHAUM utilizes breadth-first search. They utilized *auub-matrix*, *lubau* and *tubau* method to reduce the search space to decrease the runtime. However, this algorithm suffers from the stability issue and runtime fluctuation. FHAIM [29] designs the RAUL structure to store the required information and proposes two upper bounds, named EUBS and EUBR, to pruning the search space. This algorithm performs well but it is unstable and will lose the discovered itemsets in some cases. HAUIM-GMU [33] proposes the concepts of generalized maximal utility and the generalized average-utility upper bound to replace the AUUB model. It is an effective method for filtering out unpromising itemsets. But it still follows the two-phase routine that requires one more stage of candidate verification. LMHAUP [15] also proposes the concept of transaction maximum average utility upper-bound and maximum remaining average utility to replace the AUUB model. It designs the TA-List to minimize the cost for the joining process. And it shows stable complexity in both calculating upper-bounds and the mining process. The analysis of existing HAUIM algorithms are summarized in the Appendix.

2.3 High average-utility itemset mining variants

Several variants of the HAUIM problem have also been studied in the recent literature. They applied HAUIM to solve problems in dynamic and uncertain databases. We briefly review them as follows.

FUP-HAUIMD [22] is an updating algorithm, which maintain the discovered HAUIs with transaction deletion by employing the AU-list structure and modified FUP-concept. However, in the real-world, the transaction modification problem is also significant due to the some typos or input errors. In [23], an efficient framework called PRE-HAUIMI for transaction insertion in dynamic databases is developed, which relies on the AUL structures. It utilized pre-large concept, which can ensure that if the total utility in the newly inserted transaction is within the safety bound, to speed up the mining performance. However, if the number of items is huge in the newly inserted transactions, then it takes a lot of time to build the AUL-structure for the itemsets in the new transactions. This issue has a significant impact on the computational cost. APHAUIM [40] is an incremental transaction insertion algorithm based on the pre-large principle of HAUIM. For the knowledge update phase, it can significantly reduce the consumption time compared to the batch-mode approach. But during the update insertion process, it consumes more time to find the itemsets in the buffer. And another APHAUIM [41] is proposed to discover the desired possible HAUIs efficiently from uncertain datasets used in industrial IoTs. For discovering the potential itemsets of uncertainly and average-utility constraint, this algorithm is based on a level-wise model to designs the pub and $lpub$ upper bounds, and they can be used as pruning strategies to reduce the search space size. And it is performed to present the effectiveness and efficiency compared to the generic approch of HAUIM.

3 Proposed algorithm

In this section, the proposed HAUIM algorithm will be introduced in detail. It designs the KFAU-List structure, which is a novel list to capture more information about the features of items in each transaction. In additions, the relative maximum average-utility along with the pruning strategies are presented to estimate the tighter upper-bound, which can avoid the generation of unnecessary candidates during the mining process.

First, we present preliminaries related to HAUIM. Second, we introduce how to construct KFAU-List and the related definitions. And then we present the effective pruning strategies and mining approach. Finally, all the pseudo code of the proposed algorithm and the detailed explanation are given.

3.1 Preliminaries

This subsection will introduce the common definitions about the high average-utility itemsets mining methods. Let $D = \{T_1, T_2 ...., T_i\}$ be a transactional data consisting of multiple transactions, $I = \{i_1, i_2, ..., i_j\}$ be all distinct items contained in dataset D, $T = \{i_1, i_2, ..., i_k\}$ be a transaction of D that all items belong to I, $T \in D, T \subseteq I$, and $X = \{i_1, i_2, ..., i_l\}$ be an itemset and all items belong to I. Each item i has internal and external utility, which means the item quantity and unit profit, respectively.

An example transaction dataset is given in Table 1 for HAUIM. This dataset consist of seven transactions and six distinct items. Each transaction has a unique number named *Tid*. Each transaction consist of different items, which have the internal utility(i.e., quantity) of each item in the transaction. And in this paper, we don't consider the negative external utilities(i.e., profit). According to the predefined threshold of D, the HAUIM algorithm will discover all high average-utility itemsets. The common definition are presented as follows.

Definition 1 (Item Utility) The item utility of i in T is notated as $u(i, T)$ and defined as follows.

$$u(i, T) = in(i, T) \times ex(i) \tag{1}$$

Table 1 An example transaction dataset

Tid	Items in transaction
1	(b:1), (c:1), (d:2), (f:1)
2	(a:4), (b:1), (c:1), (e:1), (f:2)
3	(a:4), (b:1), (d:1), (f:2)
4	(a:7), (b:4), (e:1), (f:3)
5	(a:3), (c:1), (f:2)
6	(a:7), (d:1), (e:1), (f:1)
7	(b:3), (d:1), (f:4)

where $in(i, t)$ means the internal utility of i in T, and $ex(i)$ means external utility of i in Table 2. For example, in the given dataset, the item b in transaction T_2 is $u(b, T_2) = in(b, T_2) \times ex(b) = 1 \times 5 = 5$.

Definition 2 (Itemset Utility in a transaction) In transaction T, the utility of X is defined as follows.

$$u(X, T) = \sum_{i \in X} u(i, T). \tag{2}$$

For example, in Table 1, $u(bc, T_1) = u(b, T_1) + u(c, T_1) = 1 \times 5 + 1 \times 5 = 10$, and $u(abc, T_2) = 8 + 5 + 5 = 18$.

Definition 3 (Itemset Utility in a dataset) In dataset D, the utility of X is given by

$$u(X) = \sum_{X \subseteq T \in D} u(X, T). \tag{3}$$

For example, in Table 1, $u(ab) = u(ab, T_2) + u(ab, T_3) + u(ab, T_4) = 13+13+34 = 60$, and $u(bc) = 10+10 = 20$.

Definition 4 (Average-Utility of an itemset) In transaction T, the average-utility of an itemset X contained $l(X)$ distinct items is defined as follows.

$$au(X, T) = \frac{u(X, T)}{l(X)}. \tag{4}$$

For example, in Table 1, $au(ab, T_4) = 34/2 = 17$.

Table 2 Item profit table

Item	a	b	c	d	e	f
Profit	2	5	5	8	8	4

In dataset D, the average utility of an itemset X is given by

$$au(X) = \sum_{X \subseteq T \in D} au(X, T). \tag{5}$$

For example, in Table 1, $au(bc) = 5 + 5 = 10$.

Definition 5 (Transaction Utility) The transaction utility of a transaction T is the summation of all item utility, which is given by

$$tu(T) = \sum_{i \in T} u(i, T). \tag{6}$$

For example, the transaction utility of T_1 is $tu(T_1) = 30$, and $tu(T_2) = 34$.

Definition 6 (Total Utility) In dataset D, the total utility is given by

$$TU = \sum_{T \subseteq D} tu(T). \tag{7}$$

In the example dataset, the total utility is $TU = 30 + 34 + 29 + 54 + 19 + 34 + 39 = 239$

Definition 7 (Minimum average-utility threshold) Let θ ($0\% \leq \theta \leq 100\%$) denotes the percent value. The minimum average-utility threshold is a user-specified percentage value of total utility, which is denoted as $minAU$ and defined as

$$minAU = TU \times \theta. \tag{8}$$

For example, assume θ is 17%, then $minAU = 239 \times 17\% = 40.63$.

Definition 8 (High Average-Utility Itemset) If the utility of an itemset is no less than $minAU$, then this itemset can be termed as HAUI. For example, in Table 1, $au(bf) = 49 > minAU$. So the itemset $\{bf\}$ is HAUI.

It is a very time-consuming task to discover all HAUIs by traversing all search space of items. Therefore, the AUUB of itemset is used to reduce the search space in previous studies, which is summated the maximum utility of transaction that includes target items.

Definition 9 (Transaction Maximum Utility) The maximum utility of transaction T is given by

$$tmu(T) = \max\{u(i, T) \mid i \in T\}. \tag{9}$$

For example, in the example transaction T_2, $tmu(T_2) = \max(u(a, T_2), u(b, T_2), u(c, T_2), u(e, T_2), u(f, T_2)) = \max(8, 5, 5, 8, 8) = 8$. And in the transaction T_4, $tmu(T_4) = 20$.

Definition 10 (Average-Utility Upper-Bound) The AUUB of itemset X is defined as follows.

$$AUUB(X) = \sum_{X \subseteq T \in D} tmu(T). \quad (10)$$

For example, in Table 1, $AUUB(a) = tmu(T_2)+tmu(T_3)+tmu(T_4)+tmu(T_5)+tmu(T_6) = 8+8+20+8+14 = 58$.

Definition 11 (High Average-Utility Upper-Bound Itemset) If the AUUB of an itemset X is no less than $minAU$, then the itemset can be determined as HAUUBI.

$$HAUUBI \longleftarrow \{X \mid AUUB(X) \geq minAU\}. \quad (11)$$

Property 1 (Overestimation) In generally, the AUUB value of an itemset is no less than its actual average-utility, it means that $AUUB(X) \geq au(X)$.

Property 2 (Downward closure property) The AUUB value has downward closure property, it means that when an itemset is not HAUUBI, the itemset and its supersets will not be HAUIs.

The Properties 1 and 2 has proofed in [19]. Then the itemset whose AUUB is lower than $minAU$ can be pruned early according to the properties. These itemsets can be called unpromising itemsets.

Then we introduce the proposed HAUIM algorithm named EMAUI employing the Key feature average-utility list(KFAU-List) to capture the key features information for efficiently mining.

The algorithm constructs initial 1-itemset KFAU-Lists by scanning the given dataset twice, then recursively constructs k-itemset KFAU-Lists using 1-itemset KFAU-Lists to mine all HAUIs. The proposed algorithm does not need to verify that the generated candidate itemsets are promising, because it can calculate the actual average-utility directly based on the KFAU-Lists. For efficiently pruning the search space and avoiding generation of unpromising itemsets during the mining process, an effective pruning strategy using the tighter upper-bound of itemsets is designed. This strategy utilize the key features storing in KFAU-List to calculate the tighter upper-bound. In addition, the search space is further reduced by EA and RMUB pruning strategies. The costly unnecessary KFAU-List construction can be avoided due to the decrease of search space. According to the above optimizations, the runtime performances of our algorithm can be further improved compared to previous methods.

3.2 Constructing KFAU-List from database

The initial key feature average-utility lists of 1-itemset are constructed through the twice scanning of a given database according to the proposed algorithm.

In order to reduce the number of costly construction, the unpromising items that the AUUB value less than $minAU$ are pruned before the construction of initial KFAU-List. According to the Properties 1 and 2, the unpromising items can be determined and ignored before constructing the initial KFAU-Lists.

The algorithm scans the database for the first time and calculates the AUUB value of all items, as shown in Table 3. Then the items with $AUUB$ lower than $minAU$ do not need to build the initial KFAU-Lists.

Property 3 (Pruning the unpromising items) If the $AUUB$ of an itemset X is less than $minAU$, that is $AUUB(X) < minAU$, the itemset can be pruned before it is extended.

Proof Let itemset Y be any extension supersets of itemset X, according to the Propety 2, $AUUB(Y) \leq AUUB(X)$. Because $AUUB(X) < minAU$, then $AUUB(Y) < minAU$. And by propety 1, $au(Y) \leq AUUB(Y)$. It can be obtained $au(Y) < minAU$. □

From the Property 3 and Table 3, assuming the $minAU$ is 40, item c is unpromising item that cannot extend any HAUIs because its AUUB is 32 less than $minAU$. So it can be excluded from all transactions during the initial KFAU-List construction in the second scan. And all items are sorted in ascending order in each transaction, which is $e \prec d \prec a \prec b \prec f$, according to the AUUB of each item, as shown in Table 3.

When the second scan is completed, we can obtain the reorganized dataset, which is shown in Table 4. The item utilities are given after item identifier, which are calculated by their external and internal utilities.

The initial 1-itemset KFAU-Lists are constructed from each item present in the reorganized database. The KFAU-List of each itemset consists of multiple elements. And each element stores the key parameters that reflects the characteristics of the transaction and the itemset, in the form of (*Tid*, *iu*, *mu*, *ln*, *smu*, *rn*, *emu*). One element consists of seven fields, which are a transaction identifier(*Tid*), the actual utility of the itemset in this transaction(*iu*), the maximum item utility of items that appear after the itemset in this transaction(*mu*), the length between maximum utility item and the last item of the itemset(*ln*), the second maximum item utility that appears after the maximum utility item(*smu*), the remaining number of items that appear after itemset in the transaction(*rn*), and the tighter estimated maximum average-utility of the item superset in this transaction(*emu*). Their definitions are given as follow.

Table 3 Average-utility upper-bound of items

Item	a	b	c	d	e	f
AUUB	58	68	32	54	42	90

Table 4 Reorganized dataset

Tid	Items in tansaction			
1	d:16,	b:5,	f:4	
2	e:8,	a:8,	b:5,	f:8
3	d:8,	a:8,	b:5,	f:8
4	e:8,	a:14,	b:20,	f:12
5	a:6,	f:8		
6	e:8,	d:8,	a:14,	f:4
7	d:8,	b:15,	f:16	

Definition 12 (Maximum Item Utility of the Remaining Items) Let $X.T$ be an element in KFAU-List of X in a sorted transaction T, and item i_j be the last item of itemset X. The maximum item utility of remaining items is given by

$$X.T.mu = \max(\{u(i, T) \mid i \in T \wedge i \succ i_j\}) \quad (12)$$

For example, in Table 4, the maximum item utility of itemset d in T_1 is $\{d\}.T_1.mu = 5$, and $\{e\}.T_4.mu = 20$.

Definition 13 (Length between Maximum Utility Item with Itemset) Let $T = \{i_1, i_2, ..., i_j, i_{j+1}, ..., i_k, i_{k+1}, ..., i_N\}$ denoted a sorted transaction, i_j be the last item of X, i_k be the last item with maximum utility after the itemset X. The length between maximum utility item with the itemset in KFAU-List of X in T is defined as follows.

$$X.T.ln = k - j. \quad (13)$$

For example, in the running database, $\{e\}.T_4.ln = 2$, and $\{e\}.T_2.ln = 3$.

Definition 14 (Second Maximum Item Utility after Maximum Utility Item). Let T denote a reorganized transaction, i_k be the last item with maximum utility after X in T. The second maximum item utility after the maximum utility item in KFAU-List of X in T is defined as follows.

$$X.T.smu = max(\{u(i, T) \mid i \in T \wedge i \succ i_k\}). \quad (14)$$

For example, in Table 4, $\{e\}.T_4.smu = 12$ and $\{ed\}.T_6.smu = 4$.

Definition 15 (Number of Remaining Items) Let $T = \{i_1, i_2, ..., i_j, i_{j+1}, ..., i_k, i_{k+1}, ..., i_N\}$ denoted a reorganized transaction, i_j be the last item of X. The number of remaining items in KFAU-List of itemset X in T is given by

$$X.T.rn = N - j. \quad (15)$$

For example, in Table 4, $\{e\}.T_2.rn = 3$, and $\{eb\}.T_4.rn = 1$.

The initial KFAU-Lists of 1-itemset are constructed sequentially based on the reorganized dataset. According the ascending order of AUUBs, the KFAU-Lists of item is constructed one by one. Figure 1(a) shows initial KFAU-Lists after T_1 is completely processed, the *emu* indicates the tighter estimated maximum average-utility of the item, which is initialized to null and calculated during the mining process. The rest of transactions and items are processed and constructed in sequence, Fig. 1(b) shows the initial KFAU-Lists of all 1-itemsets.

Item f appears after other items in each revised transaction, because it has highest AUUB value among items. Therefore, all fields but Tid and iu of item f are 0.

Compared with the list structure proposed in previous studies, KFAU-List captures more information and consumes more memory for storing each itemset KFAU-List, but it can be utilized to obtain the tighter upper-bound. Take advantage of the tighter upper-bound and the corresponding pruning strategies, the costly join operation can greatly reduce, thus the total memory consumption is reduced.

3.3 Mining high average-utility itemsets with effective pruning strategies

The KFAU-Lists of k-itemsets ($k \geq 2$) can recursively generated by the initial KFAU-Lists, thus the proposed algorithm can mine HAUIs without rescanning the original dataset.

According to the KFAU-Lists of two $(k-1)$-itemsets with same prefix, the KFAU-List of k-itemset can be constructed. The combination of KFAU-List follows the ascending order of AUUB. Let itemsets pX and pY represent the itemset with same prefix p added the item i_x and i_y, respectively. According to the ascending order of AUUBs, KFAU-List of pXY can be built from KFAU-List of pX by joining pY which satisfy $i_y \succ i_x$.

The join operation of pX and pY needs to compare each element in their KFAU-Lists with the complexity of $O(m+n)$, where the m and n are the size of $|KFAU(pX)|$ and $|KFAU(pY)|$, respectively. And the m and n are large usually, because they related to the number of transactions in which the itemset pX and pY appear. So the join operation is a costly process. Meanwhile, building too many KFAU-Lists during the mining process will consume a lot of memory. Therefore it is necessary to reduce the search space as much as possible, which can minimize the number of constructed KFAU-Lists and thus improve the performance of our algorithm.

In previous studies, the maximum utility and the number of remaining items are usually used to estimate the absolute maximum average utility of the itemset, but because it is very rare for items with large utility to appear in a transaction, the estimated value of this method is usually far exceeds the maximum average utility that the superset of itemset can achieve. So we propose the relative maximum average utility upper-bound to replace the

d						
Tid	*iu*	*mu*	*ln*	*smu*	*rn*	*emu*
1	16	5	1	4	2	0

b						
Tid	*iu*	*mu*	*ln*	*smu*	*rn*	*emu*
1	5	4	1	0	1	0

f						
Tid	*iu*	*mu*	*ln*	*smu*	*rn*	*emu*
1	4	0	0	0	0	0

(a) Initial KFAU-Lists after processing T_1.

d						
Tid	*iu*	*mu*	*ln*	*smu*	*rn*	*emu*
1	16	5	1	4	2	0
3	8	8	3	0	3	0
7	8	16	2	0	2	0

e						
Tid	*iu*	*mu*	*ln*	*smu*	*rn*	*emu*
2	8	8	3	0	3	0
4	8	20	2	12	3	0
6	8	14	1	8	3	0

a						
Tid	*iu*	*mu*	*ln*	*smu*	*rn*	*emu*
2	8	8	2	0	2	0
3	8	8	2	0	2	0
4	14	20	1	12	2	0
5	6	8	1	0	1	0
6	14	8	1	4	2	0

b						
Tid	*iu*	*mu*	*ln*	*smu*	*rn*	*emu*
1	5	4	1	0	1	0
2	5	8	1	0	1	0
3	5	8	1	0	1	0
4	20	12	1	0	1	0
6	8	4	1	0	1	0
7	15	16	1	0	1	0

f						
Tid	*iu*	*mu*	*ln*	*smu*	*rn*	*emu*
1	4	0	0	0	0	0
2	8	0	0	0	0	0
3	8	0	0	0	0	0
4	12	0	0	0	0	0
5	8	0	0	0	0	0
6	4	0	0	0	0	0
7	16	0	0	0	0	0

(b) Initial KFAU-Lists after processing T_7.

Fig. 1 Initial KFAU-Lists of 1-itemsets

absolute maximum average utility upper-bound to obtain a tighter average utility estimate of an itemset. In this way, it can be judged earlier whether to construct the KFAU-List, and the unnecessary search space can be subtracted.

Definition 16 (Relative Maximum Average-utility Upper-bound) The relative maximum average-utility upper-bound for transaction T in KFAU-List of itemset X is given by.

$$rmub(X,T) = \frac{X.T.iu + X.T.mu \times X.ln}{|X| + X.T.ln} \tag{16}$$

Property 4 The X' represents any supersets of X extended by combination processes in transaction T. When the $X.T.mu > au(X,T)$, if the second maximum less than relative maximum average-utility, $X.T.smu \leq rmub(X,T)$, then we conclude $au(X',T) \leq rmub(X,T)$.

Proof Before proving property 4, we suppose that for the comparison of number $A = \frac{b+mp}{a+m}$ with number $B = \frac{b+np}{a+n}$, if $n \geq m$, we can obtain $A = \frac{b+mp}{a+m} = \frac{(b+mp)(a+n)}{(a+m)(a+n)} = \frac{\frac{(b+mp)(a+m+n-m)}{a+m}}{a+n} = \frac{b+mp+A(n-m)}{a+n}$ and $B = \frac{b+np}{a+n} = \frac{b+mp+(n-m)p}{a+n}$. Therefore, the comparison between number A and number B is essentially the comparison between number A and number p. Next multiply both numbers by $a+m$ and subtract mp, we can obtain b and ap. So, if $b \geq ap$, we conclude $A \geq B$. Proof for Property 4, let $T = \{i_1, i_2, ..., i_x, ..., i_k, ..., i_l, ..., i_n\}$, $X = \{i_1, i_2, ..., i_x\}$, $Y = \{i_p \in T \mid p > x\}$ and $X' = X \cup \{i_p \mid i_p \succ i_x \wedge i_p \in T\}$, if $X.T.mu > au(X,T)$, i_k be the maximum utility item after i_x, then the $u(i_p,T) \leq u(i_k,T) = X.T.mu$, the $au(X',T)$ can be greater than $au(X,T)$. And when the $p \leq k$, and there are at most a items that are combining with X to form X'. We can obtain $au(X',T) = \frac{u(X,T)+\sum_{i_p \in T} u(i_p)}{|X'|} \leq \frac{u(X,T)+a\times u(i_k)}{|X|+a} = P_1 = \frac{u(X,T)+a\times u(i_k)+P_1(X.ln-a)}{|X|+X.ln}$ according to the above proof. On the other hand, we can obtain $rmub(X,T) = \frac{u(X,T)+X.ln\times u(i_k)}{|X|+X.ln} = \frac{u(X,T)+a\times u(i_k)+u(i_k)\times(X.ln-a)}{|X|+X.ln}$. Because of $X.ln \geq a$, we can obtain $P_1 \leq rmub(X,T)$ through the condition $X.T.mu = u(i_k) > au(X,T) = \frac{u(X,T)}{|X|}$, finally we conclude $au(X',t) \leq rmub(X,T)$ in this case. Meanwhile, when $p > k$, $u(i_l) = X.T.smu$ and there are a items that appear before i_k, and b items that appear after i_k combine with X to form X', we can obtain $au(X',T) = \frac{u(X,T)+\sum_{i_p \in T} u(i_p)}{|X'|} \leq$

$\frac{u(X,T)+a\times u(i_k)+b\times u(i_l)}{|X|+a+b} = P_2 = \frac{u(X,T)+a\times u(i_k)+b\times u(i_l)+P_2(X.ln-a)}{|X|+X.ln+b} < \frac{u(X,T)+a\times u(i_k)+b\times u(i_l)+u(i_k)\times(X.ln-a)}{|X|+X.ln+b} = \frac{u(X,T)+X.ln\times u(i_k)+b\times u(i_l)}{|X|+X.ln+b}$ through the condition $X.T.mu = u(i_k) > au(X,T) = \frac{u(X,T)}{|X|}$ and $X.T.mu \geq X.T.smu$. In addition, we can obtain $rmub(X,T) = \frac{u(X,T)+X.ln\times u(i_k)}{|X|+X.ln} = \frac{u(X,T)+X.ln\times u(i_k)+rmub(X,T)\times b}{|X|+X.ln+b} \geq \frac{u(X,T)+X.ln\times u(i_k)+b\times u(i_l)}{|X|+X.ln+b}$ through the condition $X.T.smu \leq rmub(X,T)$, finally we conclude $au(X',T) < rmub(X,T)$ in this case. Therefore, when the $X.T.mu > au(X.T)$, if the $X.T.smu \leq rmub(X,T)$, the $rmub(X,T)$ is the upper-bound utility of $au(X',T)$. □

Due to the rarity of items with large utility, the above proof is usually established, and the $rmub(X,T)$ is usually the upper bound of average utility that X' can achieve. However, in order to ensure the correctness of the maximum average utility estimate when very few situations occur, we define the maximum average utility estimation method that apply to all situations.

Definition 17 (Tighter Estimated Maximum Average-Utility) The tighter estimated maximum average-utility for transaction T in KFAU-List of itemset X is given by.

- If $X.T.mu > X.T.iu/|X|$and $X.T.smu \leq rmub(X,T)$, then $emu(X,T) = rmub(X,T)$.
- If $X.T.mu > X.T.iu/|X|$ and $X.T.smu > rmub(X,T)$, this is a rarely situation, then $emu(X,T) = \frac{X.T.iu + X.T.mu \times X.T.ln + X.T.smu \times (X.T.rn - X.T.ln)}{|X| + X.T.rn}$
- If $0 < X.T.mu \leq X.T.iu/|X|$, then $emu(X,T) = \frac{X.T.iu + X.T.mu}{|X|+1}$.
- If $X.T.mu = 0$, then $emu(X,T) = 0$

Then in dataset D, the estimated average-utility of itemset X is given by

$$emu(X) = \sum_{X\subseteq T \wedge T\in D} emu(X,T) \tag{17}$$

It means the largest average-utility that all surpersets of X can obtain during the mining processes. According to the Definition 17, the estimated maximum average-utility of itemset X can be calculated by four conditions. In the first case, the judgment condition satisfies the Property 4, the $rmub$ is the relative maximum average-utility upper-bound for all supersets in the transaction. In case 2, this judgment condition means that the average-utility of X' can go up further, when the rest of items appear after the maximum item have the utility with $X.T.smu$ and it's larger than $rmub$. In case 3, there is no more item utility larger than $au(X,T)$, the average-utility can't be further increased. So the estimated average-utility can be calculated by $au(X,T)$ and $X.T.mu$. In case 4, it means that there is no more items appearing after X in T. Based on the estimated maximum average-utility obtained in each transaction, the highest estimated maximum average-utility of X is obtained, which is a tighter upper-bound for $au(X')$.

According to the above definitions, we not only use the relative maximum average utility to obtain a tighter upper-bound of the average utility, but also ensure the correctness of the utility estimation method in rare cases, so that a more effective pruning strategy can be obtained.

Property 5 (Pruning strategy 1) If $emu(X) < minAU$, any extension superset of X cannot be HAUI and could be pruned from the search space.

Proof Let itemset X' be the any extension supersets of X, by the Definition 17 and related statements, the $emu(X)$ is the tighter upper-bound for $au(X')$, it means $au(X') \leq emu(X)$, and because $emu(X) < minAU$, then it can be obtained $au(X') < minAU$. □

According to the pruning strategy, the unpromising itemsets can be pruned before it preforms further combination process. We can check the pruning strategy in Fig. 1. KFAU-List of d has three elements for T_1, T_3 and T_7. According to the Definition 17, the estimated maximum average-utility of d for each transaction can be calculated as follows: $emu(d,T_1) = (16+5)/(1+1) = 10.5$, $emu(d,T_3) = (8+8\times 3)/(1+3) = 8$, and $emu(d,T_7) = (8+16\times 2)/(1+2) = 13.3$. Then, $emu(d) = 10.5+8+13.3 = 31.8$. Since $emu(d)$ is less than $minAU$, which is 40, item d can be pruned before it perform the combination process.

The proposed algorithm utilize the Property 5 to overestimate the average-utility of X'. If the itemset X doesn't satisfy the Property 5, the KFAU-Lists of X' are no need to be constructed. However, the join operations is a costly process which will consume a lot of running time and memory. So in the process of constructing KFAU-List, we can terminate early when we can determine that the join operation is unnecessary. Thus, the following definition and property are employed to reduce the number of costly unnecessary join operations.

Definition 18 (Local Estimated Maximum Average-Utility) The local estimated maximum average-utility of itemset X with respect to item y ($y \succ i, \forall i \in X$) is given by

$$Lemu(X,y) = emu(X) - \sum_{X\subseteq T \wedge yT} emu(X,T). \tag{18}$$

Property 6 (Pruning strategy 2) According to the given itemsets, if $Lemu(X,y) < minAU$, the extension itemset Xy and its supersets can't be high average-utility itemsets.

Proof Let $(Xy)'$ represents the superset of Xy. According to the Definition 17 and its analysis, it can be conclude

$au[(Xy)', T] \leq emu(Xy, T) \leq emu(X, T)$. And the transactions contained X must include the transactions contained $(Xy)'$. Thus,

$$\begin{aligned} au[(Xy)'] &= \sum_{(Xy)' \subseteq T} au[(Xy)'], T \\ &\leq \sum_{(Xy)' \subseteq T} emu(Xy, T) \\ &\leq \sum_{Xy \subseteq T} emu(X, T) \\ &= emu(X) - \sum_{X \subseteq T \wedge y \nsubseteq T} emu(X, T) \\ &= Lemu(X, y) \end{aligned}$$

Therefore, if $Lemu(X, y) < minAU$, $au[(Xy)'] \leq Lemu(X, y)$, then $(Xy)' \notin HAUIs$. Similarly, if $Lemu(X, y) < minAU$, $Xy \notin HAUIs$. □

So, when the itemset X doesn't satisfy the Property 5, all items appear after X need to perform join operation. But if the Property 6 is not satisfied during the join operation, the construction of new KFAU-List can be abandoned early. The values of $emu(X)$ and $emu(X, T)$ required by the pruning strategy are stored in the KFAU-List of X.

3.4 The proposed EMAUI algorithm

In this subsection, the pseudo-codes of proposed EMAUI algorithm are presented for mining the HAUIs. For estimating the tighter upper-bound to determine whether the itemset is explored, the KFAU-List is designed and employed. Meanwhile, the tighter maximum average-utility estimation, the RMUB property and EMU strategies are presented to avoid the generation of unnecessary candidates during the mining performance. And the EA strategy is adopted for terminating the KFAU-List construction of unpromising itemsets during the costly join operation.

Algorithm 1 EMAUI algorithm.

Input: D: a transaction dataset, $minAU$: the minimum utility threshold
Output: $HAUI$: a set of high average-utility itemsets

1: Initialize an empty set to store the HAUIs, $HAUI$;
// First scan D
2: Scan D to calculate the *AUUB* of each item;
3: Let I be the list of single items sorted in *AUUB* ascending order;
// Second scan D
4: **for all** transaction T in D **do**
5: Remove any item i that $AUUB(i) < minAU$ from T;
6: Sort all items in T in ascending order by $AUUB(i)$;
7: Capture the key feature and built KFAU-Lists for each item $i \in I$;
8: **end for**
9: **Mining**$(null, I, minAU)$;
10: **return** $HAUI$;

Algorithm 1 shows the main procedure of EMAUI. It initializes an empty set $HAUI$ to store the high average-utility itemsets (Line 1). Then, the algorithm first scans the dataset to calculate the $AUUB$ of each item (Line 2). The procedure sorts single items in ascending order by $AUUB$ and stores them in list I(Line 3). Next, in order to construct initial KFAU-Lists, the algorithm second scans the dataset, removes the unpromising items and captures the key features (Lines 4-8). Then, the initial KFAU-Lists of each item i has stored in the list I. By calling *Mining*, the recursive mining procedure is performed to generate high average-utility itemsets, which are stored in $HAUI$(Line 9). Finally, all HAUIs are output (Line 10).

Algorithm 2 Mining.

Input: p: the prefix itemset, pEx: a list of extensions of p, $minAU$: the minimum utility threshold
Output: $HAUI$: the set of high average-utility itemsets

1: **for** *each itemset* $X \in pEx$ **do**
2: $pX \longleftarrow \{p \bigcup X\}$
3: **if** $au(pX) \geq minAU$ **then**
4: $HAUI \longleftarrow \{HAUI \bigcup pX\}$;
5: **end if**
6: Call the **EMU calculation** procedure;
7: **if** $emu(pX) \geq minAU$ **then**
8: Let $pXEx$ be the list of extensions of pX, and $pXEx \longleftarrow \varnothing$
9: **for** *each item* $y \in pEx$ *and* $y \succ X$ **do**
10: $py \longleftarrow \{p \bigcup y\}$
11: $KFAUL(pXy) \longleftarrow$ **KFAU-List Construction**(p, pX, py);
12: **if** $KFAUL(pXy) \neq \varnothing$ **then**
13: $pXEx \longleftarrow \{pXEx \bigcup pXy\}$;
14: **end if**
15: **end for**
16: **Mining**$(pX, pXEx, minAU)$
17: **end if**
18: **end for**
19: **return** $HAUI$;

Algorithm 2 shows the *Mining* pseudo-code of EMAUI. For each itemset X in pEx, if the average-utility of pX is no less than $minAU$, pX can be determined as HAUI (Lines 1-5). Then, the estimated maximum average-utility of pX is calculated by calling function *EMU Calculation* algorithm (Line 6). If its value is no less than $minAU$, the extensions of pX will be considered to further recursive mining (Lines 7-17). Each item y in pEx such that $y \succ X$ will try to add to pX for constructing the KFAU-Lists of pXy (Lines 9-11). However, due to the EA pruning strategy applied in the *KFAU-List Construction* algorithm, the KFAU-List of pXy needs to be determined to be non-empty, and then

pXy is added to the extension itemsets of pX (Lines 12-14). Then, the *Mining* program is called with pX and $pXEx$ recursively (Line 16). When the procedure terminates, all HAUIs have been completely discovered.

Algorithm 3 EMU calculation.

Input: $KFAUL(X)$: the KFAU-List of itemset X
Output: $emu(X)$: the estimated maximum average-utility of X

1: $emu(X) = 0$;
2: **for** *each element* $E_X \in KFAUL(X)$ **do**
3: $T = E_X.tid$;
4: **if** $X.T.mu \neq 0$ **then**
5: **if** $X.T.mu > au(X)$ **then**
6: $X.T.emu = rmub(X, T)$;
7: **if** $X.T.smu > X.T.emu$ **then**
8: $X.T.emu = \frac{X.T.iu+X.T.mu\times X.T.ln+X.T.smu\times(X.T.rn-X.T.ln)}{|X|+X.T.rn}$;
9: **end if**
10: **else**
11: $X.T.emu = \frac{X.T.iu+X.T.mu}{|X|+1}$;
12: **end if**
13: **else**
14: $X.T.emu = 0$;
15: **end if**
16: $emu(X) = emu(X) + X.T.emu$;
17: **end for**
18: **return** $emu(X)$;

The pseudo-code of the *EMU Calculation* algorithm is presented as Algorithm 1, which is called in line 6 in *Mining* procedure. The algorithm calculates $emu(X)$ by accumulating the $emu(X, E_X.tid)$ of each element in KFAU-List of X (Lines 2-17). Each $emu(X, T)$ is calculated in four cases according to the Definition 17. During the calculation, each value of $emu(X, T)$ is stored in the corresponding element of KFAU-List, and will be utilized in the EA strategy during the construction.

Algorithm 4 shows the KFAU-List construction procedure of EMAUI. The variable *emu* is initialized as $emu(X)$, which is prepared for the EA strategy (Line 2). Because all elements in KFAU-List are sorted by Tid, the common transaction that appears in two KFAU-Lists can be determined by comparing the Tids in the two lists (Lines 3-18). The *emu* value will decrease by the $emu(X, T)$ value stored in the KFAU-List of pX, when $KFAUL(py)$ does not have the element with same Tid as $KFAUL(pX)$ (Lines 9-10). If the *emu* decreases lower than $minAU$, it means that the Property 6 can't satisfied. Then the construction of pXy KFAU-List can be terminated and there is no need to explore any supersets of pXy (Line 11-12). This strategy avoids unnecessary join operation and reduces runtime cost and memory usage. Note that the construction of k-itemsets ($k > 2$) KFAU-List, the iu of each element in KFAU-List of pXy needs to subtract the iu of the corresponding element in KFAU-List of prefix p (Lines 19-29). Finally, the KFAU-List of pXy is returned to *Mining* procedure (Line 30).

Algorithm 4 KFAU-List Construction.

Input: $KFAUL(p)$: the KFAU-List of itemset p; $KFAUL(pX)$: the KFAU-List of itemset pX; $KFAUL(py)$: the KFAU-List of itemset py;
Output: $KFAUL(pXy)$: the KFAU-List of itemset pXy;

1: $KFAUL(pXy) = \varnothing$; Let $i, j = 0$;
2: Let $emu = emu(pX)$; // For the**EA strategy**
3: **while** $i < KFAUL(pX).size$ *and* $j < KFAUL(py).size$ **do**
4: $E_X = KFAUL(pX)[i]$; $E_y = KFAUL(py)[j]$;
5: **if** $E_X.Tid == E_y.Tid$ **then**
6: $E_{Xy} \longleftarrow < E_X.Tid, E_X.iu + E_y.iu, E_y.mu, E_y.ln, E_y.smu, E_y.rn >$;
7: i=i+1; j=j+1;
8: $KFAUL(pXy) \longleftarrow KFAUL(pXy) \bigcup E_{Xy}$;
9: **else if** $E_X.Tid < E_y.Tid$ **then**
10: $emu = emu - E_X.emu$;
11: **if** $emu < minAU$ **then**
12: **return** $\varnothing$; // EA strategy
13: **end if**
14: i=i+1;
15: **else**
16: j=j+1;
17: **end if**
18: **end while**
// For constructing k-itemsets KFAU-Lists ($k > 2$)
19: **if** $KFAUL(p) = \varnothing$ and $KFAUL(pXy) = \varnothing$ **then**
20: Let $i, j = 0$;
21: **while** $i < KFAUL(pXy).size$ *and* $j < KFAUL(p).size$ **do**
22: $E_{pXy} = KFAUL(pXy)[i]$; $E_p = KFAUL(p)[j]$;
23: **if** $E_{pXy}.Tid = E_p.Tid$ **then**
24: $E_{pXy}.iu = E_{pXy}.iu - E_p.iu$;
25: i=i+1;
26: **end if**
27: j=j+1;
28: **end while**
29: **end if**
30: **return** $KFAUL(pXy)$;

Table 5 Characteristics of the datasets

Dataset	$\|D\|$	$\|I\|$	$\|T\|$	*maxLen*	Size	Type
Chain	1112949	46087	7.22	170	79.2 MB	Sparse
Pumsb	49046	2113	74	74	25.1 MB	Sparse
Accidents	340183	468	33.8	51	63.1 MB	Dense
Mushroom	8124	119	23	23	1.03 MB	Dense
Chess	3196	75	37	37	641 KB	Dense
Retail	88162	16470	10.3	76	6.5 MB	Sparse

4 Experimental evaluation

4.1 Experimental settings

In this section, the performance of EMAUI is evaluated by comparing to the state-of-art list-based HAUI algorithms, which are MHAI [45], HAUI-Miner [19], HAUIM-GMU [33] and FHAUM [21].

In this section, we evaluate the performance of the proposed algorithm. The performance of EMAUI is compared to the state-of-art MHAI, HAUI-Miner, HAUIM-GMU and FHAUM algorithms, which all are list-based HAUI algorithms. We use a computer with 3.7 GHz Ryzen 9 5900X AMD Processor, 64GB RAM, and Windows 10 operating system to perform all experiments.

All running time, memory consumption, number of join operation, and scalability experiments are performed with six standard datasets chain, pumsb, accidents, mushroom, chess, and retail, which are download from the SPMF repository [5]. All five algorithms are tested on different datasets using the Java programming language. The characteristics of each dataset are given in Table 5. Where $|D|$ represents the total number of transactions, $|I|$ represents the total number of distinct items, $|T|$ represents the average number of items per transaction, *maxLen* represents the maximum number of items included in a transaction.

Chain dataset is real-life customer transaction dataset with real utility values. Others datasets are real datasets but with synthetic utility values, the internal utility values are generated by a uniform distribution in [1, 10]. They are

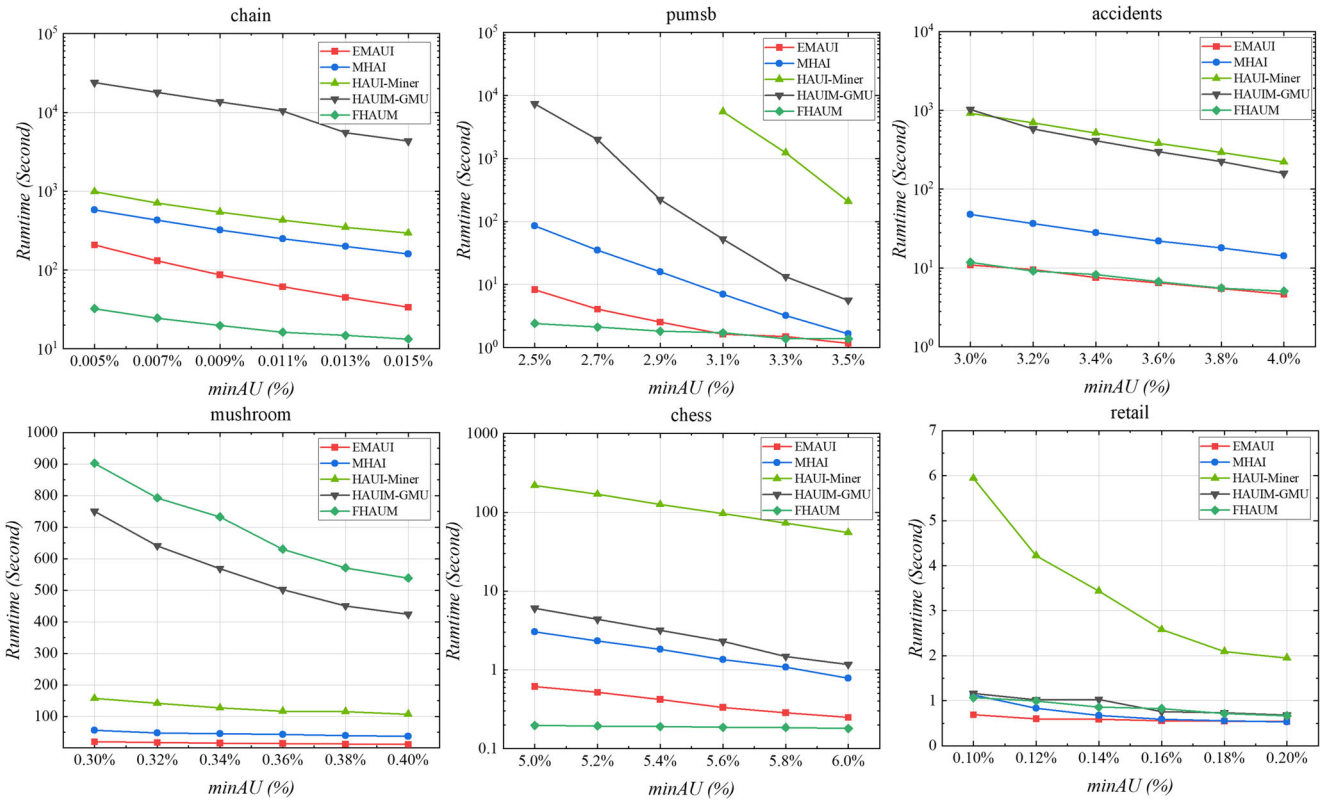

Fig. 2 Runtime comparison

all widely used in the performance evaluation of HAUIM algorithms.

4.2 Runtime performance evaluation

In this subsection, the Runtime performance evaluation of algorithms which conducted on six real datasets are shown in Fig. 2. All tests were performed by gradually increasing $minAU$. As the $minAU$ increases, the running time of all algorithms decrease. It is because that the number of HAUIs discovered from datasets decrease as $minAU$ become higher. The first result is the runtime performance for chain dataset where $minAU$ values are set between 0.005% to 0.015%. It can be observed that the running time of EMAUI is lower than those of MAHI, HAUI-Miner and HAUM-GMU for all values of $minAU$. The FHAUM is faster than other algorithms, but it misses a lot of HAUIs. According to the calculation, it misses 3.5% to 6.1% of the number of HAUIs compared to the other algorithms within the selected $minAU$ range. And when the $minAU$ is larger, it misses more number of HAUIs. All algorithms have a consistent downward trend, and the runtime used by EMAUI decreases the fastest as the $minAU$ increases. The experiment result of pumsb dataset is shown as the second graph in Fig. 2. It was performed by setting $minAU$ between 2.5% and 3.5%. It should be noticed that the running time of HAUI-Miner at 2.5% and 2.9% were not given because the HAUI-Miner performs the mining more than 12 hours. When the $minAU$ is increased to 3.5%, the EMAUI execution time is less than other algorithms, even the FHAUM. It is because that there isn't high average-utility itemset, the EMAUI has more efficient pruning strategies than other algorithms for candidates.

The third to fifth graphs in Fig. 2 present the results of runtime experiments on three dense datasets which are accidents, mushroom and chess. The experiments are performed on accidents, mushroom and chess datasets by using the $minAU$ settings, which are 3.0%-4.0%, 0.3%-0.4% and 5.0%-6.0%, respectively. The EMAUI is faster than all other algorithms on accidents and mushroom datasets, and slightly slower than FHAUM on chess dataset. Meanwhile, we observed that FHAUM will miss more HAUIs on dense datasets, at least 10% and up to 49% at 3.0% $minAU$ on accidents dataset. While the EMAUI algorithm always has a low runtime and does not miss any HAUIs.

For the retail dataset, the time consumed by each algorithm are not much different except for HAUI-Miner, as shown in the last graph in Fig. 2. Even so, when the $minAU$ is set as 0.1%, the EMAUI is more than 1. 55 times faster than other algorithms. It is because that the size of retail dataset is small and it is a sparse dataset. The number of transactions contained in an itemset is fewer than others, it is fast to construct a new utility-list. When the $minAU$ is small, the time of first and second scan execution will be the most part of the total execution time, which leads to that the improvement of pruning strategy only slightly decrease the runtime of the algorithm.

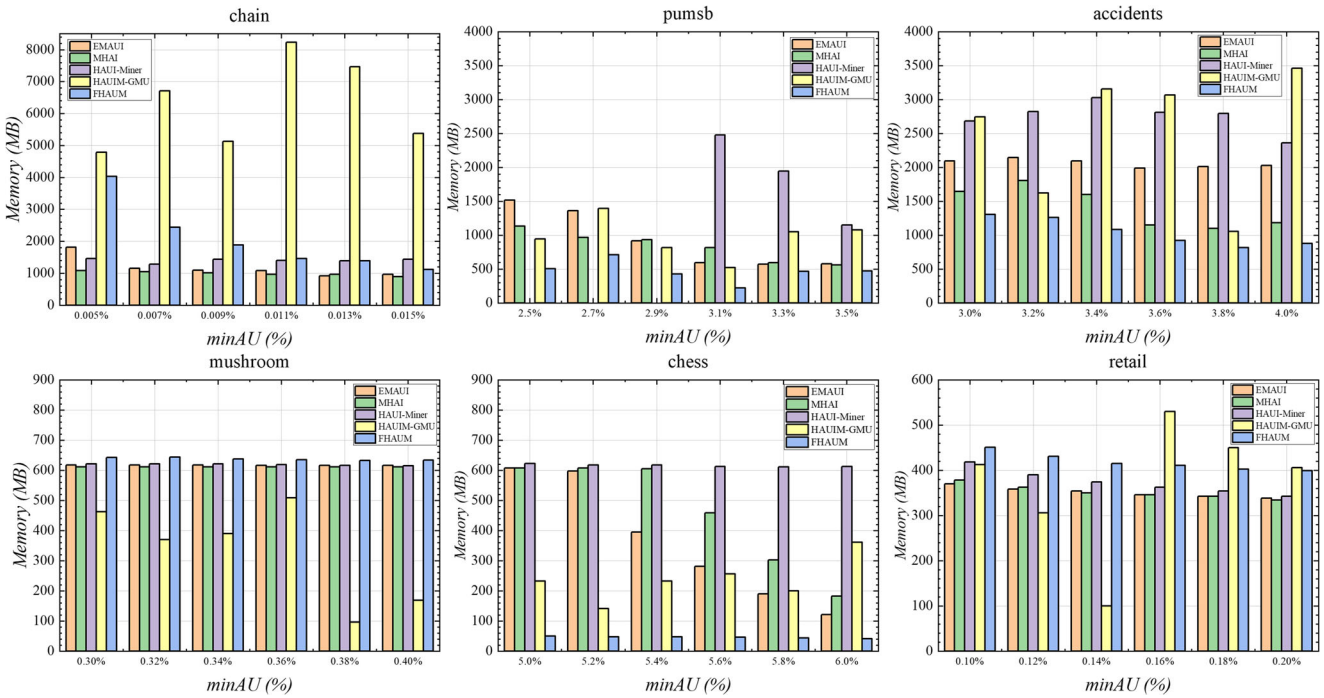

Fig. 3 Memory usage

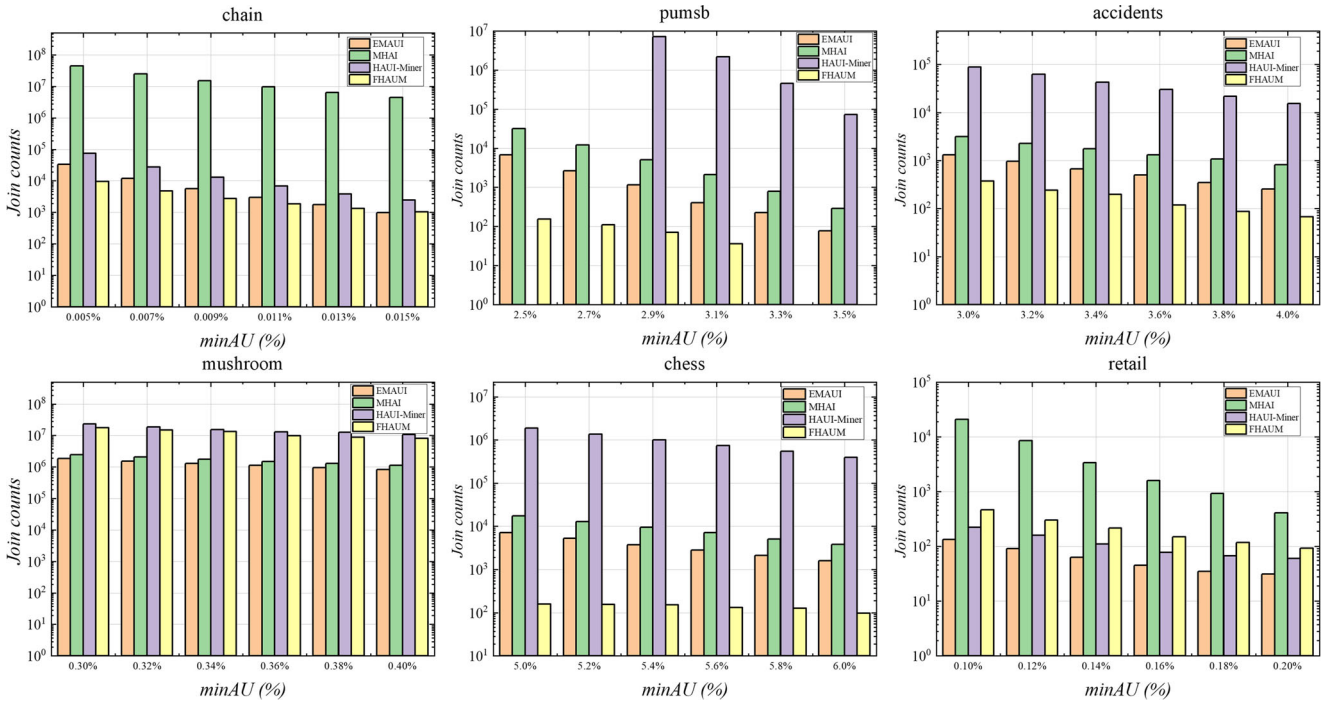

Fig. 4 Number of join operation

According to the experimental results on various characteristic datasets, it can be concluded that EMAUI consistently outperforms other comparison algorithms for most datasets under the premise of ensuring the correct number of HAUIs mined. Meanwhile, the proposed algorithm has good stability under both sparse and dense datasets, and has a faster runtime decrease speed when the $minAU$ is increased.

4.3 Memory performance evaluation

In this subsection, the memory performances of algorithms which performed on six datasets are shown in Fig. 3. We use the standard Java API to measure the peak memory consumption during program operation. The memory experiments have the same $minAU$ setting as runtime experiments. As shown in Fig. 3, the memory consumption of algorithms gradually decrease as the $minAU$ increases. It is because that as the value of $minAU$ increase, the potential candidate itemsets is decreased.

When the $minAU$ is large, it can be observed that the memory performance of EMAUI is not much different from other algorithms in most cases. It is because that the increased memory consumption of the new fields is comparable to the memory saved by reducing the KFAU-List construction. And in some cases, the EMAUI uses less memory than other algorithms due to the number of candidate nodes being considered is significantly reduced. Also, although FHAUM saves a lot of memory consumption in individual cases, it misses a lot of HAUIs, so the memory savings is moot. Therefore, the experimental results show that the memory performance of EMAUI is not much different from other algorithms when the $minAU$ is large, even though the EMAUI consumes more memory to capture more features in each element.

4.4 Number of join operations

In this subsection, the join count performance of algorithms, except the HAUIM-GMU algorithm, are shown in Fig. 4, which have the same $minAU$ and run on the same dataset as previous experiments. Because the HAUIM-GMU is a Two-Phase mining algorithm, it doesn't make sense to count the number of its join operation. According to the average-utility lists of (k-1)-itemsets, all other four list-based algorithms perform join operation to construct the average-utility lists of k-itemsets. Because the join operation needs to compare each element in the two (k-1)-itemset list, it requires more running time and computational cost. Therefore, the fewer the number of join operation performed, the better the performance of the algorithm. According to the Fig. 4, the number of join operation gradually decrease as the $minAU$ increases. On most datasets, the MHAI generates lower number of candidates than HAUI-Miner because its tight pruning strategy. But in the first and last graphs of Fig. 4, the join counts of HAUI-Miner are less than that of MHAI for chain and retail datasets. It is because that the HAUI-Miner

Table 6 Evaluation of the RMUB and EA strategies on chain and retail datasets

Dataset/ $minAU$	$EqRmub\ counts$/ $NEqRmub\ counts$	$Rmub\%$	$EA\ counts$/ $Join\ counts$	$EA\%$
chain				
0.015%	3483427 / 357575	90.69%	3457253 / 980	99.97%
0.013%	3750878 / 385995	90.67%	4987150 /1746	99.97%
0.011%	4048754 / 417768	90.65%	7890314 / 2967	99.96%
0.009%	4445016 / 462354	90.58%	12216301 / 5580	99.95%
0.007%	4941452 / 516452	90.54%	20717130 / 11899	99.94%
0.005%	5611641 / 591452	90.47%	38457722 / 33895	99.91%
retail				
0.20%	104997 / 8813	92.26%	235 / 31	88.35%
0.18%	115263 / 10126	91.92%	488 / 35	93.31%
0.16%	129993 / 11860	91.64%	989 /45	95.65%
0.14%	147122 / 14009	91.31%	1727 / 63	96.48%
0.12%	166814 / 16512	90.99%	4114 / 91	97.84%
0.10%	194090 / 19950	90.68%	10084 / 133	98.70%

algorithm recalculates the max utility of transaction during the database revising, which could exclude unpromising candidate on specific sparse datasets.

For all datasets, it is obvious that the EMAUI performs fewer join operations than MHAI and HAUI-Miner for various $minAU$. This is because that the estimated maximum average-utility upper-bound adopted by EMAUI is tighter than those of MHAI and HAUI-Miner. In addition to the EA strategy adopted in EMAUI, the proposed algorithm further reduce the unnecessary join operations during the KFAU-List construction. Although the join counts of FHAUM is smaller than that of EMAUI, they obtained by skipping many undiscovered HAUIs, and such improvement of pruning efficiency is not desirable. Even so, the FHAUM still underperforms the EMAUI on some datasets. So it can be seen that the join count performance of EMAUI has been significantly improved on all datasets in comparison with other algorithms. Because of the significant reduction in the number of join operations, the algorithm has good performance in terms of memory consumption.

4.5 Evaluation of the RMUB and EA strategies

The EMAUI algorithm adopts RMUB and EA strategies to reduce the upper-bound and KFAU-Lists construction times, respectively, thus reducing the search space and saving running time. In order to evaluate the efficiency of RMUB and EA strategies, we used the same parameter $minAU$ as the previous experiments on chain and retail datasets to conduct the experiment.

In order to evaluate the efficiency of the RMUB strategy, we recorded the number of **successes** of the RMUB strategy($EqRmub\ counts$) and the number of **failures** of the RMUB strategy($NEqRmub\ counts$), and then calculated the ratio of the RMUB effective to the total number of judgements($Rmub\% = EqRmubcounts/(EqRmub\ counts + NEqRmub\ counts)$) to obtain the RMUB pruning efficiency. This ratio tells how much the upper bounds of the itemsets are reduced by the RMUB strategy. Meanwhile, the number of Join operations reduced by EA strategy($EA\ counts$) and the number of actual join operations after EA reduction($Join\ counts$) have been recorded. And then we calculated the ratio of the number of Join operations reduced by EA to the number of Join operations without the EA strategy($EA\% = EA\ counts/(EA\ counts + Join\ counts)$). This ratio shows how many unnecessary construction operations can be stopped by EA strategy.

The results are shown in Table 6. In this experiment, the higher the ratios, the more effective the strategies.

As we can see, for two real customer transaction datasets, the $Rmub$ strategy can be used as the tighter estimated maximum average utility over 90% of the time when estimating the upper bound of utility. Moreover, with the increase of $minAU$, the probability of successes of the $Rmub$ strategy also slightly increases. The results indicate that $Rmub$ strategy can effectively replace the the absolute maximum average utility upper-bound to estimate the more tighter maximum average utility of itemset, which greatly reduces the upper-bounds and thus reduces the search spaces. Meanwhile, such a high $Rmub$ ratio also illustrate

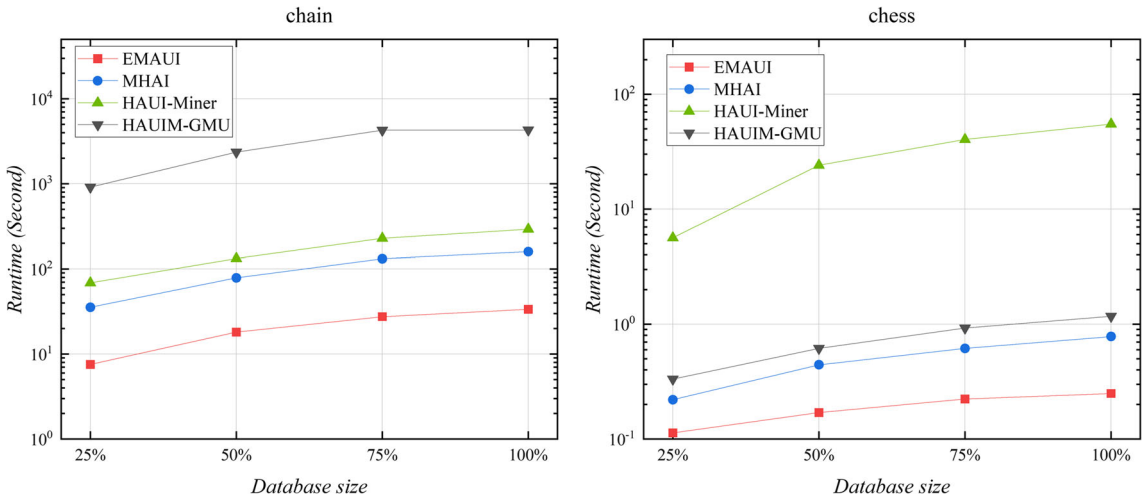

Fig. 5 Scalability experiments

that it is rare for multiple itemset with large utility to appear at a same transaction. So in order to prevent the failure of *Rmub* strategy, it is sufficient to record the second greatest utility, there is no need to record more greatest utility values. In addition, the results of $EA\%$ show that EA strategy can greatly reduce hopeless KFAU-Lists constructions because EA adopts a tighter information of local estimated maximum average-utility to prune the search spaces. On the chain and retail datasets, EA can terminate more than 99% and 88% of the list constructions early, respectively. These two datasets can represent the common sparse datasets in practical applications. Therefore, it is very necessary to adopt RMUB and EA strategies, they can obtain good performance gains, and can exert effects in real applications.

4.6 Scalability performance evaluation

In the last set of experiments, since FHAUM will miss many HAUIs, the scalability performance of the other four algorithms are evaluated on two typical datasets with different types. By changing the dataset size in 25% increments of the number of transactions, the runtime performance of all algorithms is studied. The chain and chess datasets were experimented with a fixed $minAU$ setting of 0.015% and 6%, respectively. The results of experiments are presented in Fig. 5. It can be seen that the running time of algorithms gradually increase as size of the dataset increases on both dense and sparse datasets. Meanwhile, it can be observed that the EMAUI has better performance than other algorithms regardless of the dataset size and type. In addition, compared with other algorithms, the running time consumption of EMAUI increases more slowly. So it can be concluded from the experiment that the scalability performance of EMAUI is better than other comparison algorithms.

4.7 Discussion

Based on the KFAU-List structure that captures more key features information of each itemset, the estimated maximum average utility upper-bound is significantly reduced using the RMUB strategy. In order to further avoid the generation of unnecessary costly join operation, the EA pruning strategies are utilized in this paper. The presented algorithm is evaluated on six benchmark datasets. The results of experiments are quite useful and more actionable.

The following poins can further improve our algorithm. In order to capture more key features information, each item KFAU-List requires more field to store. It leads to that the memory consumption of total KFAU-Lists is more than before. Although the new pruning strategies have reduced the total number of the KFAU-Lists, but the maximal memory consumption will still be higher than MHAI in many cases. That is because when the $minAU$ is small, the summation of new field in all KFAU-Lists consumes more memory than the reduced number of KFAU-List construction. Therefore, consider using data compression in KFAU-List, the memory performance of our algorithm can be further enhanced.

With the rapid development of data mining technology, studying the application of HAUIM algorithm in big data, and designing the cloud compution version of HAUIM algorithm to mine HAUIs in Big data have a very broad application scenario. As similar to pHAUI-Miner [28], EMAUI can easily be redesign as a distributed algorithm to mine HAUIs in Big data. Since modern processors contain multiple cores that can process multiple tasks at the

same time, we can apply multi-threaded parallel processing during iterative mining process. In detail, we can divide the search space into multiple sub-search spaces according to items, and then each thread iteratively mine HAUIs in the sub-search spaces using a divide-and-conquer strategy. In order to maintain the load balance of each thread, the sub-search space can be divided according to the number of extension supersets. Thus, this will significantly reduce the runtime to mine all HAUIs. In the future, we can study how to use multi-threads approach to mine HAUIs in cloud computing, such as MapReduce framework.

5 Conclusions

In this paper, a novel KFAU-List based algorithm named EMAUI is presents for mining HAUIs. KFAU-List is designed to capture more key features information of each item in each transaction, thus the tighter upper-bound is estimated by the RMUB definition. According to the tighter RMUB and EA strategies, EMAUI algorithm prunes the search space and avoids the generation of unnecessary candidates greatly. Various experiments with six benchmark datasets, evaluating the the runtime, memory, join counts, RMUB and EA efficiency, and scalability performance of EMAUI algorithm. From the results of experiments, it can be seen that the EMAUI algorithm has good or better performance than the existing list-based HAUIM algorithms.

The design of list, tighter estimating method, and several pruning strategies introduced in this paper can be applied to other HAUIM tasks such as incremental mining, sequential pattern mining and stream data mining. In additions, an interesting HUIM field is top-k HUI mining, which could mining the HUIs without setting the minimal utility threshold. Considering such mining tasks, we plan to conduct research on the top-k HAUIM algorithm on the basis of the proposed algorithm. Other possible applications are also scheduled to be explored.

Acknowledgments This research was supported by the Natural Science Foundation of China (61172080, 61771357).

Appendix

Table 7 The summary about the existed HAUIM algorithms

Algorithm	Data Structure	Pruning Strategy	Base Algorithm	Year	Pros and cons
HAUP-growth	HAUP-tree	*auub*	Apriori	2010	This algorithm shows better performances than HAU which is based on the Apriori-like approach. However, it needs additional arrays to to keep the data used by the mining process. It generates lots of unpromising candidates and consumes excessive time for mining process.
TPAU		*auub*	Apriori	2011	TPAU performs better than two-phase approach in terms of number of candidates and execution time. Horeover, the efficiency of this algorithm is significantly decreased, when the candidates increases abruptly. Moreover, it incurs scalability issues as well.
PBAU	Index Table	HAUUBI	Apriori	2012	PBAU algorithm with pruning strategies performs better than traditional TPAU approach in terms of execution time and memory consumption. However, the performance is not measured when the dataset is not fitted into memory. The partition techniques are not implemented on the dataset to effectively achieve the mining task.
HAUI-tree	Index Table	HAUUBI	Apriori	2014	According to the revised tree structure, it optimizes the memory requirement and improves the calculating speed of itemsets values. However, when the dataset increases, the memory usuage of HAUI-tree is considerable increases.
HAUI-Miner	AU-list	TMUDC	HUI-Miner	2016	It is firstly algorithm that explores the search space without candiate generation according to the average-utility list. The experimental results shows that HAUI-Miner perform well than the state-of-the-art algorithms, HAUP-Growth, PAI and HAUI-tree. However, it performs too many costly join operations to mine the HAUIs.

Table 7 (continued)

Algorithm	Data Structure	Pruning Strategy	Base Algorithm	Year	Pros and cons
EHAUPM	Modified AUlist	lub, $rtub$, ECAUPS and SUJ pruning	HAUI-Miner	2017	This algorithm designs many pruning strategies to accelerate the mining speed based on the HAUI-Miner. However, the algorithm have runtime fluctuation and is less stable. And the proposed two upper bounds are not able to efficiently handle the large dataset.
MHAI	High Average-Utility Itemset list	$auub$ and mau	HAUI-Miner	2017	It offers a novel list structure HAI-list to capture mor information. And it can estimate the average-utility of the super itemsets before the branch of itemset is mined. However, the sorting technique for constructing list structure is not efficient enough.
FHAUM	AU-list	$auub-Matrix$, $lubau$, $tubau$	HAUI-Miner	2017	Experimental results show that the proposed algorithms, D-FHAUM and B-FHAUM show better performance than the state-of-the-art algorithms PAI, HAUI-tree and HAUI-Miner. However, it suffers from the stability issue and run-time fluctuation.
dHAUIM	IDUL	Depth Pruning, Width Pruning, Strong Width Pruning	HAUI-Miner	2018	The proposed algorithm outperforms the benchmark HAUIM algorithms in terms of execution time and number of join operations. But it consumes a lot of computational resources to obtain the utility vector of the item set. And it performs too many costly join operations in case of large datasets.
HAUL-Growth	HAUL-Tree and IL	eub, $teub$, $bteub$ and $max\text{-}reub_k$	HAUI-Miner	2019	This algorithm significantly performs better than dHAUIM and TUB-HAUPM in terms of execution time, memory usage, number of join operations and scalability.
VMHAUI	NVUV-list	$vmsub1$, $vmsub$, $vmaub$ and $vtopmaub$	HAUI-Miner	2019	VMHAUI outperforms the state-of-art-algorithms, MHAI, TUB-HAUPM and dHAUIM in terms of run-time, memory usage and number of join operations.
FHAIM	RAUL	$EUBS$, $EUBR$	HAUI-Miner	2020	This algorithm utilizes the RAUL sturcture, EUBS and EUBR upper bounds to prune the search space. It performs well but is not stable and will lose the discovered itemsets in some cases.
HAUIM-GMU	ItemInfo Table	$GAUUB$, Support pruning strategy	Apriori	2021	HAUIM-GMU algorithm outperforms existing state-of-the-art algorithms, HAUI-Tree, HAUI-Miner and EHAUPM. But it still follows the two-phase routine that requires one more stage of candidate verification.
LMHAUP	TA-List	$tmaub$, $mrau$	HAUI-Miner	2021	LMHAUP utilizes the TA-List sturcture, tmaub and mrau upper bounds to prune the search space.It shows stable complexity in both calculating upper-bounds and the mining process.

References

1. Agrawal R, Srikant R (1994) Fast algorithms for mining association rules. In: Proceedings 20th International Conference. Very Large Data Bases, VLDB. pp 487–499
2. Ahmed CF, Tanbeer SK, Jeong B et al (2009) Efficient tree structures for high utility pattern mining in incremental databases. IEEE Trans on Knowl Data Eng 21(12):1708–1721
3. Bouasker S, Ben Yahia S (2015) Key correlation mining by simultaneous monotone and anti-monotone constraints checking. In: Proceedings of the 30th Annual ACM Symposium on Applied Computing. pp 851–856
4. Djenouri Y, Belhadi A, Fournier-Viger P et al (2018) Fast and effective cluster-based information retrieval using frequent closed itemsets. Inf Sci 453:154–167
5. Fournier-Viger P, Gomariz A, Gueniche T et al (2014a) Spmf: A java open-source pattern mining library. http://www.philippe-fournier-viger.com/spmf/index.php
6. Fournier-Viger P, Wu CW, Zida S et al (2014b) Fhm: Faster high-utility itemset mining using estimated utility co-occurrence pruning. In: Andreasen T, Christiansen H, Cubero JC, et al (eds) Foundations of Intelligent Systems. Springer International Publishing. pp 83–92
7. Fournier-Viger P, Lin JCW, Kiran RU et al (2017) A survey of sequential pattern mining. Data Sci Pattern Recognit 1(1):54–77
8. Fournier-Viger P, Lin J, Nkambou R et al (2019a) High-Utility Pattern Mining: Theory, Algorithms and Applications. Springer International Publishing
9. Fournier-Viger P, Zhang Y, Chun-Wei Lin J et al (2019b) Mining local and peak high utility itemsets. Inf Sci 481:344–367

10. Gan W, Chun-Wei J, Chao H et al (2018) Coupm: Correlated utility-based pattern mining. In: 2018 IEEE International Conference on Big Data (Big Data). pp 2607–2616
11. Gan W, Lin JCW, Fournier-Viger P et al (2019) A survey of parallel sequential pattern mining. ACM Trans Knowl Discov Data 13(3):1–34
12. Han J, Pei J, YJAsr Yin (2000) Mining frequent patterns without candidate generation. ACM SIGMOD Record 29(2):1–12
13. Han J, Pei J, Kamber M (2011) Data mining: concepts and techniques. Elsevier
14. Hong TP, Lee CH, Wang SL (2011) Effective utility mining with the measure of average utility. Expert Syst Appl 38(7):8259–8265
15. Kim H, Yun U, Baek Y et al (2021) Efficient list based mining of high average utility patterns with maximum average pruning strategies. Inf Sci 543:85–105
16. Krishnamoorthy S (2017) Hminer: Efficiently mining high utility itemsets. Expert Syst Appl 90:168–183
17. Lan GC, Hong TP, Tseng VSJJoIs et al (2012) A projection-based approach for discovering high average-utility itemsets. J Inf Sci Eng 28(1):193–209
18. Lin CW, Hong TP, Lu WH (2010) Efficiently mining high average utility itemsets with a tree structure. In: Asian conference on intelligent information and database systems. Springer. pp 131–139
19. Lin JCW, Li T, Fournier-Viger P et al (2016) An efficient algorithm to mine high average-utility itemsets. Adv Eng Inform 30(2):233–243
20. Lin JCW, Ren S, Fournier-Viger P et al (2017a) Ehaupm: Efficient high average-utility pattern mining with tighter upper bounds. IEEE Access 5:12,927–12,940
21. Lin JCW, Ren S, Fournier-Viger P et al (2017b) A fast algorithm for mining high average-utility itemsets. Appl Intell 47(2):331–346
22. Lin JCW, Shao Y, Fournier-Viger P et al (2018) Maintenance algorithm for high average-utility itemsets with transaction deletion. Appl Intell 48(10):3691–3706
23. Lin JCW, Pirouz M, Djenouri Y et al (2020) Incrementally updating the high average-utility patterns with pre-large concept. Applied Intelligence
24. Liu M, Qu J (2012) Mining high utility itemsets without candidate generation. In: Proceedings of the 21st ACM international conference on Information and knowledge management. pp 55–64
25. Liu Y, Wk Liao, Choudhary A (2005) A two-phase algorithm for fast discovery of high utility itemsets. In: Pacific-asia conference on knowledge discovery and data mining. Springer berlin heidelberg, advances in knowledge discovery and data mining. pp 689–695
26. Lu T, Vo B, Nguyen HT et al (2015) A new method for mining high average utility itemsets. In: IFIP international conference on computer information systems and industrial management, Springer. pp 33–42
27. Nguyen LTT, Vu VV, Lam MTH et al (2019) An efficient method for mining high utility closed itemsets. Inf Sci 495:78–99
28. Nguyen LTT, Nguyen TD, Nguyen A et al (2020) Efficient method for mining high-utility itemsets using high-average utility measure. In: International conference on computational collective intelligence, Springer. pp 305–315
29. Sethi KK, Ramesh D (2020) A fast high average-utility itemset mining with efficient tighter upper bounds and novel list structure. The journal of supercomputing
30. Shin SJ, Lee DS, Lee WSJIS (2014) Cp-tree: an adaptive synopsis structure for compressing frequent itemsets over online data streams. Inf Sci 278:559–576
31. Singh K, Shakya HK, Singh A et al (2018) Mining of high-utility itemsets with negative utility. Expert Systems 35(6):e12–296
32. Singh K, Kumar A, Singh SS et al (2019) Ehnl: an efficient algorithm for mining high utility itemsets with negative utility value and length constraints. Inf Sci 484:44–70
33. Song W, Liu l, Huang C (2021) Generalized maximal utility for mining high average-utility itemsets. Knowledge and Information Systems
34. Szathmary L, Napoli A, Valtchev P (2007) Towards rare itemset mining. In: 19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007). vol 1. IEEE, pp 305–312
35. Truong T, Duong H, Le B et al (2019a) Efficient vertical mining of high average-utility itemsets based on novel upper-bounds. IEEE Trans Knowl Data Eng 31(2):301–314
36. Truong T, Duong H, Le B et al (2019b) Efficient high average-utility itemset mining using novel vertical weak upper-bounds. Knowledge-Based Systems 183:104–847
37. Tseng VS, Wu CW, Shie BE et al (2010) Up-growth: an efficient algorithm for high utility itemset mining. In: Proceedings of the 16th ACM SIGKDD international conference on knowledge discovery and data mining. pp 253–262
38. Tseng VS, Shie B, Wu C et al (2013) Efficient algorithms for mining high utility itemsets from transactional databases. IEEE Trans Knowl Data Eng 25(8):1772–1786
39. Uno T, Kiyomi M, Arimura H (2004) Lcm ver 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In: Fimi
40. Wu JMT, Teng Q, Lin JCW et al (2020) Updating high average-utility itemsets with pre-large concept. J Intell Fuzzy Syst 38(5):5831–5840
41. Wu JMT, Li Z, Srivastava G et al (2021) Analytics of high average-utility patterns in the industrial internet of things. Applied Intelligence
42. Yao H, Hamilton HJ, Butz CJ (2004) A foundational approach to mining itemset utilities from databases. In: Proceedings of the 2004 SIAM international conference on data mining. SIAM. pp 482–486
43. Yen SJ, Lee YS, Wu CW et al (2009) An efficient algorithm for maintaining frequent closed itemsets over data stream. In: International conference on industrial, engineering and other applications of applied intelligent systems. Springer. pp 767–776
44. Yildirim I, Celik M (2019) An efficient tree-based algorithm for mining high average-utility itemset. IEEE Access 7:144,245–144,263
45. Yun U, Kim D (2017) Mining of high average-utility itemsets using novel list structure and pruning strategy. Futur Gener Comput Syst 68:346–360
46. Yun U, Nam H, Kim J et al (2020) Efficient transaction deleting approach of pre-large based high utility pattern mining in dynamic databases. Futur Gener Comput Syst 103:58–78
47. Zaki MJ (2000) Scalable algorithms for association mining. IEEE Trans on Knowl and Data Eng 12(3):372–390
48. Zida S, Fournier-Viger P, Lin JCW et al (2015) Efim: a highly efficient algorithm for high-utility itemset mining. In: Mexican international conference on artificial intelligence. Springer international publishing, advances in Artificial Intelligence and Soft Computing. pp 530–546

Gufeng Li received the B.E. degree in communication engineering in 2015 and M.E. degree in software engineering in 2018 from Xidian University. Now he is working toward the PhD degree in Artificial intelligence and modern communications at Xidian University. His current research interests include data mining, artificial intelligence, and artificial intelligence communication.

Tao Shang received his BE and ME degrees from Huazhong University of Science and Technology (Wuhan, China) in 1994 and 2001, respectively, and his PhD degree from Shanghai Jiao Tong University in 2006. From 1994 to 1997, he worked at the Institute of Optics and Electronic, Chinese Academy of Sciences. Now, he is a professor at the State Key Laboratory of Integrated Service Network, School of Telecommunication Engineering, Xidian University (Xian, China). His main research topic covers big data processing, artificial intelligence and modern communications, and marine information sensing technology.

Yinling Zhang received the B.Eng. degree in communication engineering from Shandong University, Jinan, Shandong, China, in 2015 and the M.E. degree in Telecommunications Engineering at Xidian University, Xi'an, Shaanxi, China in 2018. Now she is working toward the PhD degree in Artificial intelligence and modern communications at Xidian University. Her research interests include data fusion and marine information sensing.