MEMU: More Efficient Algorithm to Mine High Average-Utility Patterns with Multiple Minimum Average-Utility Thresholds

Article *in* IEEE Access · February 2018

DOI: 10.1109/ACCESS.2018.2801261

3 authors, including:

Chun-Wei Jerry Lin
Høgskulen på Vestlandet
252 PUBLICATIONS **1,718** CITATIONS

SEE PROFILE

Philippe Fournier Viger
Harbin Institute of Technology (Shenzhen)
216 PUBLICATIONS **2,027** CITATIONS

SEE PROFILE

MEMU: More Efficient Algorithm to Mine High Average-Utility Patterns with Multiple Minimum Average-Utility Thresholds

Jerry Chun-Wei Lin[1,2], *Member, IEEE*, Shifeng Ren[2], and Philippe Fournier-Viger[3]
[1]Department of Computing, Mathematics, and Physics, Norway University of Applied Sciences, Bergen, Norway
[2]School of Computer Science and Technology, Harbin Institute of Technology Shenzhen Graduate School, Shenzhen, China
[3]School of Humanities and Social Sciences, Harbin Institute of Technology Shenzhen Graduate School, Shenzhen, China

High average-utility itemsets mining (HAUIM) is an emerging topic in data mining. Compared to traditional high utility itemset mining, HAUIM more fairly measures the utility of itemsets by considering their lengths (number of items). Many previous studies have presented algorithms to efficiently mine high average-utility itemsets (HAUIs). Most of these algorithms, however, only mine HAUIs using a single minimum high average-utility threshold, which limits their usefulness to analyze real data. This is a problem because different items are not equally important to the user. The importance of an item can be expressed for example in terms of weights, interestingness or unit profit. In the past, a baseline algorithm called HAUIM-MMAU was presented to mine HAUIs using multiple minimum high average-utility thresholds. However, it uses a generate-and-test approach to mine HAUIs using a level-wise approach, which is time consuming. In this paper, we propose an efficient algorithm to discover HAUIs based on the average-utility list structure. A tighter upper-bound model is used to reduce the search space instead of the one used in traditional HAUIM, which is called the *auub* model. Three pruning strategies are also respectively developed to increase the performance HAUIs. Experiments show that the proposed algorithm outperforms the state-of-the-art HAUIM-MMAU algorithm in terms of runtime, memory usage, number of candidates and scalability.

***Index Terms*—data mining; high average-utility itemsets; list structure; multiple thresholds.**

I. Introduction

The goal of data mining is to reveal interesting, important and useful information from databases that satisfy the requirements of different applications. Various types of data mining tasks can be performed on a database to extract useful patterns or models such as association rule mining (ARM) [1, 11], sequential pattern mining (SPM) [2, 35], high-utility itemset mining (HUIM) [4, 7, 18, 30, 41, 44], classification [6, 40], and clustering [3]. Apriori [1] and FP-growth [11] are the most well-known algorithms for mining association rules (ARs) and frequent itemsets (FIs) in databases. Several extensions of pattern mining have been studied such as incremental mining [4, 39] and constraint-based mining [19, 27, 36, 42, 43].

FIM and ARM are useful but only consider the occurrence frequencies of items in a database to evaluate patterns. Some other implicit factors such as unit profits of items, importance, or interestingness of patterns are not considered in traditional ARM. Thus, patterns are discovered while ignoring some aspects that can be important to users. To solve this problem, the task of high-utility itemset mining (HUIM) [37, 44, 45] was presented to mine the set of high-utility itemsets, in which an itemset is a high-utility itemset if its utility is no less than a user-defined minimum high-utility threshold (count). Since the traditional downward closure (DC) property of FIM does not hold in HUIM for the utility measure, a transaction-weighted utility (TWU) model [18] was designed. This model alllows to obtain a variation of the DC property, called transaction-weighted downward closure (TWDC) property. Using that property and by maintaining a set of high transaction-weighted utilization itemsets (HTWUIs), the search space can be reduced to cope with the "computational explosion" of the search space for low minimum utility threshold values. An improved IIDS [20] strategy was also presented to restore the anti-monotonicity property and improve mining performance. Tseng et al. [41] proposed several pruning strategies and the utility-pattern (UP)-tree structure for speeding up the mining performance.

The above algorithms, however, suffer the problem of high memory complexity since they keep candidate high utility itemsets in memory. To avoid this problem, several algorithms [22, 30] were presented to mine HUIs without candidate generation. These were shown to have better performance than those based on the traditional TWU model. However, previous studies suffer from an important limitation. It is that each itemset is only evaluated using a single minimum high utility threshold, which results in an unfair evaluation since items in a database may have different properties, importance or weights. Hence, using a single minimum high-utility threshold may lead to the "rare item problem". For instance, the itemset $\{diamond\}$ may be regarded as a HUI in a chain store if its profit is no less than 5,000$, while the itemset $\{milk, bread\}$ may be considered as a HUI if its profit is no less than 100$. This is because sales of diamond should be evaluated differently than those of milk and bread, as they have different importance for retaillers. Using a single minimum high-utility threshold may cause two problems: 1) If the minimum high-utility threshold is set too high, few itemsets are discovered. 2) If the minimum high-utility threshold is set too low, a very large number of meaningless itemsets may be discovered. High-utility itemset mining with multiple minimum high-utility threshold (HUIM-MMU) [27] was proposed to let the user assign a minimum utility threshold for each item.

In recent decades, HUIM has been extensively studied and used to address the requirements of various applications.

Corresponding author: Jerry Chun-Wei Lin (email: jerrylin@ieee.org).

However, HUIM still suffers from the intrinsic drawback that long itemsets tend to have a higher utility than shorter ones. Thus, the utility measure may provide an unfair measurement for short itemsets. To more fairly evaluate itemsets, high average-utility itemset mining (HAUIM) was proposed [13], which considers the size of each itemset (number of items) as well as its utility to determine if it is interesting to the user. An itemset is considered to be a high average-utility itemset (HAUI) if its average-utility value is greater than a minimum high-average utility threshold (count) set by the user. To obtain a downward closure property for mining HAUIs and use it to efficiently mine HAUIs, a new average-utility upper-bound (*auub*) model [13] was designed. Thereafter, most HAUIM algorithms [21, 24, 28] have used this model to reduce the search space and improve the performance of HAUIM. However, these algorithms evaluate whether an itemset is a HAUI using a single minimum high average-utility threshold. To deal with this problem, a first algorithm called high average-utility itemset mining with multiple minimum average-utility thresholds (HAUIM-MMAU) [29] was presented. It uses a generate-and-test and level-wise approach to mine HAUIs. Although this algorithm is correct and complete, it suffers from intrinsic drawbacks of Apriori-like approaches, which are to perform multiple database scans and consider a huge number of unpromising candidates. These drawbacks may lead to poor mining performance. To improve the performance of HAUIM, this paper presents an algorithm based on an efficient list-based structure to mine HAUIs without candidate generation and without performing multiple database scans. A new upper-bound model is used instead of the traditional *auub* model and three efficient pruning strategies are developed to reduce the search space, as well as eliminate many unpromising itemsets. Major contributions of this study are summarized as follows.

1) A more efficient algorithm is proposed to discover all HAUIs using multiple minimum high-average utility thresholds based on the average-utility (AU)-list framework. Each item can be associated with a user-defined minimum high average-utility threshold, which is more realistic than the original HAUIM task.
2) An improved upper-bound model, a sorted enumeration tree to represent the search space for mining HAUIs, and three pruning strategies are respectively developed to reduce the search space by eliminating unpromising itemsets.
3) Extensive experiments are conducted on several real-world and synthetic datasets to compare the performance of the proposed algorithm with the state-of-the-art algorithm in terms of runtime, memory usage, number of candidates and scalability.

The rest of the paper is organized as follows. Related work is discussed in Section II. Preliminaries and problem statement are given in Section III. The proposed sorted enumeration tree, the improved upper bound, the proposed algorithm, and three pruning strategies are respectively introduced in Section IV. An illustrated example to explain the designed algorithms step-by-step is presented in Section V. Extensive experiments on several datasets are provided in Section VI. A discussion and a conclusion are provided in Section VII.

II. Related Work

Association rule mining (ARM) and frequent itemset mining (FIM) are fundamental tasks in data mining, which have been widely studied and applied in various domains [1, 11, 19, 36, 43]. The well-known Apriori algorithm was initially proposed [1] to mine association rules (ARs) in two phases. In the first phase, the set of frequent itemsets (FIs) are discovered in a level-wise manner using a user-defined minimum support threshold. In the second phase, FIs having a support no less than the threshold are combined to form ARs, and those having a confidence greater than a user-defined minimum confidence threshold are shown to the user. The Apriori algorithm is known to generate a very large number of itemsets and perform numerous database scans to evaluate their support (occurrence frequencies). To speed up the mining performance of FIs, a compact tree structure called FP-tree was designed. This structure is constructed by scanning a database once. A mining algorithm named FP-growth [11] was proposed to recursively discover all FIs using the FP-tree structure [11]. Other well-known algorithms are Eclat [46] and dEclat [47]. Those algorithms rely on a vertical database representation to efficiently mine all FIs without performing multiple database scans. A drawback of previous studies is, however, that a single minimum support threshold is used to evaluate patterns. This results in an unfair evaluation of patterns since each item in a database has different properties, such as weight, interestingness, or importance. Hence, items should not be treated equally. To consider differences between items in terms of their occurrence frequencies, FIM with multiple minimum support thresholds was proposed, and an algorithm named MSApriori was designed to efficiently discover FIs for this task [17]. Users of MSApriori have to specify a minimum item support threshold for each item. MSApriori evaluates itemsets based on the thresholds of items that they contain. Thanks to this approach, the MSApriori algorithm can discover rules containing rare items and ignore many meaningless rules containing frequent items. However, the MSApriori algorithm still suffers from drawbacks of the Apriori algorithm, which are to perform many database scans and consider a large number of candidates during the mining phase. To address this performance issue, the CFP-growth algorithm was proposed [12]. It discovers all FIs using a pattern growth method using a structure called MIS-tree. To further improve the performance of CFPGrowth, CFP-growth++ was proposed with several pruning strategies. In particular, it employs the least minimum support (LMS) instead of the minimum MIS value of each item in a database to reduce the search space.

An important issue with the task of FIM is that it does not consider important information about items in database such as their purchase quantities in transactions and their unit profits. Thus, FIM cannot satisfy the requirements of users who are interested in discovering itemsets that yield a high profit. To address this problem of traditional FIM, high utility itemset mining (HUIM) was proposed [16, 38, 39, 44]. An important challenge in traditional HUIM is the "combination explosion"

of the search space for discovering the set of high-utility itemsets (HUIs). To solve this issue, the first algorithm called transaction-weighted utility (TWU) model [18] was proposed based on the Apriori framework. The TWU algorithm relies on a transaction-weighted downward closure (TWDC) property of high transaction-weighted utilization itemsets (HTWUIs) to reduce the search space. The TWDC property ensures the correctness and the completeness of the algorithm to discover high-utility itemsets (HUIs) from the set of HTWUIs. To improve the performance of HUIM by reducing the number of candidate itemsets, the IIDS algorithm [20] was proposed based on the TWU model. The designed IIDS strategy can be applied to all level-wise utility mining methods to reduce the number of candidates that they consider. In each pass, a IIDS-based HUIM algorithm scans a database that is smaller than the original database since isolated items are skipped to improve the performance. However, Apriori-like approaches have to perform multiple database scans to discover all HUIs. To discover HUIs more efficiently, a tree-based compact high utility pattern (HUP)-tree structure and its corresponding mining procedure were presented. The HUP-tree algorithm stores the purchase quantities of items in tree nodes, thus speeding up the mining performance to discover HUIs. The UP-growth and UP-growth+ algorithm [41] also introduced additional search space pruning strategies based on a designed UP-tree structure, which can be built by performing two database scans. These algorithms maintain information about HUIs in the UP-tree structure to speed up the mining process. It was shown that pruning strategies developed in UP-Growth and UP-Growth+ [41] can efficiently reduce the search space and the number of candidates. However, the above algorithms can still generate a huge number of candidates, and many candidate itemsets are kept in memory during the first phase, which degrades the mining performance. Thus, several one-phase algorithms [22, 30] were respectively proposed to find HUIs without candidate generation. Liu et al. [30] presented a novel utility-list structure to quickly calculate the utility of any itemset and upper-bounds on the utility of its extensions using the simple intersection operation. This utility-list structure keeps only the information that is relevant from the database to mine HUIs. In [22], the authors presented a pattern-growth approach that search HUIs using a reverse set enumeration tree and prune the search space using various upper-bounds. Also, a developed linear data structure was used to compute a tight bound for pruning and directly identifying high utility patterns efficiently. Experiments have shown that one-phase algorithms have better performance compared to two-phase algorithms in most cases.

The above studies focus on mining HUIs by assuming that unit profits of items are positive. However, in a real-life transaction database, items may have negative weights/unit profits. For example, if a customer buys three units of an item (A) in a supermarket, (s)he may receive a unit of item (B) for free as a promotion to promote product (B). Suppose that each unit of item (A) yields a profit of five dollars, and each unit of item (B) that is given away costs two dollars. Although giving away an unit of the item (B) results in a loss of two dollars for the supermarket, selling three units of (A) that are cross-promoted with item (B) generates 15 dollars. Chu et al. [5] have first studied the problem of mining HUIs with negative unit profit values. The developed HUINIV-Mine algorithm extends the TWU model to cope with negative unit profit values to mine HUIs efficiently. Lan et al. [25] then proposed a three-scan mining algorithm to mine high on-shelf utility itemsets with negative profit values from temporal databases. Moreover, an effective itemset generation method was developed to avoid generating a large number of redundant candidates and to effectively reduce the number of data scans to find itemsets. Line et al. [8] also presented a matrix to keep the relationships between 2-itemsets in memory, thus speeding up HUI mining performance by considering the negative value.

The above studies focused on improving the efficiency for discovering all HUIs using a single minimum high utility threshold. They do not consider the nature of each item, which can lead to the "rare pattern problem". Lin et al. respectively proposed the HUI-MMU algorithm [27] and HIMU algorithm [10] to mine HUIs using Apriori-like and list-based approaches. Those two algorithms can automatically identify the minimum utility threshold to be used for each item using an approach inspired from MISApriori. The proposed model is more realistic than the traditional problem of HUIM since it can treat each item differently. Several algorithms were designed and used in various applications [31, 33].

A major drawback of traditional HUIM is that it only evaluates patterns based on their utilities, and this measurement is often unfair for short itemsets. To reveal better patterns by considering their lengths, high average-utility itemset mining (HAUIM) [13] was proposed. It provides a more fair utility measure called the average utility. An itemset is said to be a high average-utility itemset (HAUI) if its utility divided by its size (number of items) is no less than a user-defined minimum average-utility threshold. The first algorithm for HAUIM is the two-phase TPAU algorithm [13]. It introduced an average-utility upper bound (*auub*) model to estimate the utilities of patterns in the search space. The *aaub* model allows to safely reduce the search space, while ensuring the completeness and correctness of the algorithm for HAUIM. Since the TPAU algorithm is a level-wise, it suffers from limitations of Apriori-based algorithms. To improve the performance for HAUIM, the projection-based PAI algorithm [23] was developed with a novel pruning strategy. The high average-utility pattern (HAUP)-tree structure and a mining algorithm called HAUP-growth [21] were designed to overcome the problem of performing multiple database scans. Each node of a HAUP-tree stores the purchase quantities of its prefix items. Thus, HAUIs can be directly derived from this tree. This approach is efficient but can consume a large amount of memory for databases containing long transactions. Lu et al. presented the HAUI-tree approach [24] to mine HAUIs based on an index table. Using this approach, the number of candidates for mining HAUIs can be greatly reduced. Besides, a new itemset structure was developed to improve the speed for calculating the values of itemsets and reduce memory usage. A novel more efficient HAUI-Miner algorithm [28] was developed with two pruning strategies to mine HAUIs based on a compact list structure. The average-utility (AU)-list structure was designed

to keep relevant information for mining HAUIs. However, previous studies do not consider the nature of each item in a database as they use a single minimum high average utility threshold. To obtain a more fair measurement of the utility of itemsets, Lin et al. [29] proposed an Apriori-like algorithm, named HAUIM-MMAU, which discover all HAUIs using multiple minimum high average-utility thresholds. However, HAUIM-MMAU suffers from the same drawbacks as other Apriori-based algorithms. It performs multiple database scans to discover all HAUIs from a huge number of candidates that are kept in memory. Thus, it is quite challenging and critically important to design more efficient algorithms to improve the performance of HAUI mining with multiple minimum high average-utility thresholds. Several algorithms were proposed and HAUIM with multiple thresholds is an active research area [32, 34].

III. Preliminaries and Problem Statement

Let $I = \{i_1, i_2, \ldots, i_m\}$ be a finite set with m distinct items. A quantitative database is a set of transactions $D = \{T_1, T_2, \ldots, T_n\}$, in which each transaction $T_q \in D$ $(1 \leq q \leq m)$ is a subset of I and has a unique identifier q, called its *TID*. Besides, each item i_j in a transaction T_q, has a purchase quantity (defined as a positive integer), and denoted as $q(i_j, T_q)$. A profit table ptable = $\{p(i_1), p(i_2), \ldots, p(i_m)\}$ shows the profit value of each item i_j. A set of k distinct items $X = \{i_1, i_2, \ldots, i_k\}$ such that $X \subseteq I$ is said to be a k-itemset, in which k is the length of the itemset. An itemset X is considered to be contained in a transaction T_q if $X \subseteq T_q$.

A quantitative database is shown in Table I. It will be used as a running example to illustrate the proposed approach step-by-step. The database has six items denoted from (*a*) to (*f*). A profit table is shown in Table II, which is used to show the unit profit of each item in the database. Note that only positive unit profits are considered in this paper for items. Some studies in HUIM have previously proposed approaches to handle negative unit profits [5, 8, 25]. Extending the proposed approach presented in this paper for negative unit profits will be considered for future work.

The task of high average-utility itemset mining (HAUIM) was proposed as a variation of the task of high-utility itemset mining (HUIM). The definitions of HUIM can be found in [44, 45]. The definitions related to HAUIM are presented in the following paragraphs.

TABLE I: A sample database.

TID	transaction (item:quantity)
1	*a*:1, *c*:10, *e*:5
2	*b*:20, *c*:6, *d*:2, *e*:3
3	*a*:1, *b*:10, *c*:4, *d*:2
4	*a*:3, *c*:3, *d*:3
5	*c*:5, *e*:2
6	*a*:2, *b*:10, *d*:3, *e*:5, *f*:11

TABLE II: A unit profit table.

Item	***a***	***b***	***c***	***d***	***e***	***f***
Profit	7	2	3	4	3	2

Definition 1 (Item utility)**.** The utility of an item i_j in a transaction T_q is denoted as $u(i_j, T_q)$, and defined as:

$$u(i_j, T_q) = q(i_j, T_q) \times p(i_j), \tag{1}$$

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

For example in Table I, the utility of (a) in transaction T_1 is calculated as $u(a)$ (= 1× 7) (= 7).

Definition 2 (Average-utility of an item in a transaction)**.** The average-utility of an item (i_j) in a transaction T_q is denoted as $au(i_j, T_q)$, and defined as:

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

For example in Table I, the average-utility of the item (a) in transaction T_1 is calculated as $au(a, T_1) = \frac{7}{1}$ (= 7), which is equal to its utility value in traditional HUIM.

From the above definition, it can be found that the average-utility of a single item is no different than its utility in HUIM [18, 44]. The reason is that the size of an itemset containing a single item is 1. By dividing its utility by 1, it is found that the average-utility of an item is equal to its utility in HUIM.

Definition 3 (Average-utility of an itemset in a transaction)**.** The average-utility of a k-itemset X in a transaction T_q is denoted as $au(X, T_q)$, and defined as:

$$au(X, T_q) = \frac{\sum_{i_j \in X \wedge X \subseteq T_q} u(i_j, T_q)}{|X| = k}. \tag{3}$$

For example in Table I, the average-utility of the itemset (ab) in transaction T_3 is calculated as $au(ab, T_3) = \frac{7+20}{2}$ (= 13.5).

Definition 4 (Average-utility of an itemset in *D*)**.** The average-utility of an itemset X in a database *D* is denoted as $au(X)$, and defined as:

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

For example in Table I, the average-utility of the itemset (ac) is calculated as: $au(ac)$ (= 13.5 + 9.5 + 15)(= 38).

Definition 5 (Multiple minimum high average-utility counts)**.** The minimum high average-utility threshold of an item i_j in a database D is denoted as $mau(i_j)$. A special data structure named *MAU-Table* stores the set of minimum high average-utility counts corresponding to all items in *D*. This table is defined as:

$$\textit{MAU-Table} = \{mau(i_1), mau(i_2), \ldots, mau(i_r)\}, \tag{5}$$

where $1 \leq r \leq m$.

For example, the minimum high average-utility count (*mau*) of each item in the illustrated example can be defined as in Table III.

Definition 6 (Least minimum high average-utility count, LMAU)**.** The least minimum high average-utility count is the

TABLE III: The MAU-Table.

Item	***a***	***b***	***c***	***d***	***e***	***f***
mau	80	41	42	75	68	41

minimum *mau* value for items in *D*, which is denoted as *LMAU* and defined as:

$$LMAU = min\{mau(i_1), mau(i_2), \ldots, mau(i_r)\}. \quad (6)$$

In the illustrated example, the *LMAU* is calculated as *LMAU* = *min*{80, 41, 42, 75, 68, 41} (= 41). The *LMAU* value is a lower-bound on the *mau* values of discovered itemsets. An itemset cannot be a HAUI if its average-utility is less than the *LMAU*. Thus, the *LMAU* plays an important role for pruning unpromising itemsets based on the *auub* model or the proposed *rtub* model, which will be later explained in section IV.

Definition 7 (Minimum high average-utility count of an itemset)**.** The minimum high average-utility count of a *k*-itemset *X* in *D* is denoted as $mau(X)$, and defined as:

$$mau(X) = \frac{\sum_{i_j \in X} mau(i_j)}{|X|(= k)} \quad (7)$$

In the running example, the $mau(ab)$ (=$\frac{80+41}{2}$)(= 60.5), and $mau(abc)$(= $\frac{80+41+42}{3}$)(= 54.3).

In the single minimum high average-utility mining framework, the average-utility upper bound (*auub*) model was designed to provide the transaction-maximum utility downward closure (TMUDC) property of high average-utility upper-bound itemsets (HAUUBIs), which can be used to reduce the search space. Definitions are given below.

Definition 8 (Transaction-maximum utility, *tmu*)**.** The transaction-maximum utility of a transaction T_q is denoted as $tmu(T_q)$, and defined as:

$$tmu(T_q) = max\{u(i_j, T_q)|i_j \subseteq T_q\}. \quad (8)$$

For example, $tmu(T_1)$ is calculated as $tmu(T_1)$ = *max*{7, 30 15} (= 30), which is an upper bound for any itemset (*X*) contained in transaction T_1.

Based on the transaction-maximum utility, an average-utility upper-bound (*auub*) of the itemsets can be obtained for mining HAUIs. It is defined as follows.

Definition 9 (Average-utility upper bound of an itemset in *D*, *auub*)**.** The average-utility upper bound of an itemset X in a database D is denoted as $auub(X)$, and defined as:

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

For example in Table I, the average-utility upper bound of itemset (ac) in the database is calculated as: $auub(ac)$ (= 30 + 20 + 12) (= 62), which is no less than the minimum high average-utility count ($>$ 41).

Theorem 1 (Anti-monotonicity property of *auub*)**.** *Let Y be the extension of an itemset X such that $X \subseteq Y$. The following relationship holds:*

$$auub(Y) \leq auub(X). \quad (10)$$

Proof. Let the set of transaction IDs of itemsets X and Y be denoted as $tids(X)$ and $tids(Y)$, respectively. Because $X \subseteq Y$, it follows that $tids(Y) \subseteq tids(X)$. Thus, we can obtain that:
$auub(Y) = \sum_{T_q \in tids(Y)} tmu(T_q) \leq \sum_{Tq \in tids(X)} tmu(T_q) = auub(X)$. □

Based on this *auub* model, all the supersets (child nodes) of an itemset *X* can be directly pruned if $auub(X) < LMAU$ or $auub(X)$ is less than the minimum high average-utility count in the traditional HAUI mining framework (using a single threshold). Thus, a large part of the search space can be pruned. This property has been used in previous studies to limit the exploration of the search space for mining HAUIs. Based on the *auub* model, the transaction-maximum utility downward closure (TMUDC) property was designed to maintain a small set of high average-utility upper-bound itemsets (HAUUBI). It is defined as follows.

Definition 10 (High average-utility upper-bound itemset, *HAUUBI*)**.** An itemset X is called a high average-utility upper bound itemset (HAUUBI) if its average-utility upper bound is no less than the minimum high average-utility count, which is defined as:

$$HAUUBI \leftarrow \{X|auub(X) \geq mau(X)\}. \quad (11)$$

For example in Table I, the itemset (ab) is not considered as a HAUUBI since its average-utility upper bound is $auub(ab)$ = $(40 < 60.5)$.

Theorem 2 (HAUIs $\subseteq$ HAUUBIs)**.** *The TMUDC property ensures that HAUIs $\subseteq$ HAUUBIs. Hence, if an itemset is not a HAUUBI, then none of its supersets will be HAUIs.*

Problem Statement: An itemset X is considered as a HAUI iff its average-utility is no less than the minimum high average-utility count, that is:

$$HAUI \leftarrow \{X|au(X) \geq mau(X)\}. \quad (12)$$

IV. Proposed Algorithm for Mining HAUIs

In the past, an Apriori-like HAUIM-MMAU [29] algorithm was first proposed to discover all HAUIs with multiple minimum high average-utility counts. However, HAUIM-MMAU is a generate-and-test approach. Thus it suffers from the same drawbacks as the Apriori algorithm (generating and maintaining many candidates in memory, as well as scanning the database numerous times). To improve the mining performance, a more efficient algorithm with multiple minimum high average-utility counts (called **MEMU**) is proposed in this paper. It efficiently discovers HAUIs without candidate generation and without perfomring multiple database scans. A sorted enumeration tree and a tight upper-bound model are presented to speed up the mining performance of the proposed algorithm and reduce the search space for mining HAUIs. Details are given below.

A. Proposed sorted enumeration tree and tighter upper-bound model

Prior studies [13, 21, 24, 28] mainly focused on discovering HAUIs with a single minimum average-utility threshold based on the average-utility upper-bound (*auub*). This model alllows to reduce the search space based on the transaction-maximum utility downward closure (TMUDC) property. However, this downward closure property does not hold when considering multiple minimum average-utility thresholds. The reason is that each discovered itemset has a unique minimum average-utility threshold and thus this property cannot guarantee that all supersets of an itemset *X* will have a lower minimum average-utility threshold than that of $mau(X)$, which may lead to not discovering the complete set of HAUIs. To restore the pruning ability of the TMUDC property for the case of multiple minimum high average-utility thresholds, a sorting strategy is developed based on the use of a sorted enumeration tree for exploring the search space of HAUIs. Details are provided next.

Definition 11 (Sorting strategy)**.** Items in the MAU-Table are sorted according to the ascending order $\prec$ of minimum high average-utility thresholds.

In the illustrated example, the sorted order is $b \prec c \prec e \prec d \prec a$. Thus, items in the enumeration tree are also sorted to maintain the completeness and correctness of the algorithm for discovering HAUIs. For example, the sorted enumeration tree of item (*b*) in the illustrated example is shown in Fig. 1. When searching for itemsets, each new itemset is obtained by combining the items suceeding the current one according to the sorting order. For example, the itemset (*bce*) is generated by combining the itemsets (*bc*) and (*be*) through their transaction IDs.

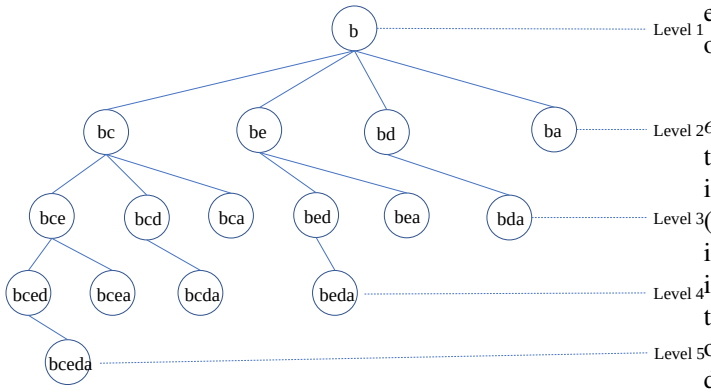

Fig. 1: Sorted enumeration (SE)-tree of the item (*b*).

Property 1. *Let Y be an extension (superset) of X in the SE-tree such that $X \subseteq Y$. The relationship $mau(Y) \geq mau(X)$ holds.*

Proof. Let $Y \backslash X$ denotes the remaining items in Y such that $(Y \backslash X = Y - X)$. Without loss of generality, consider that all items in the itemsets *Y* and *X* are sorted and it is known that $X \subseteq Y$. Thus, we can obtain that:

$$mau(X) = \frac{\sum_{i_j \in X} mau(i_j)}{|X|} < mau(Y) = \frac{\sum_{i_j \in X} mau(i_j) + \sum_{i_j \in Y \setminus X} mau(i_j)}{|Y|}.$$ □

In the running example, $mau(bce)$(= 50.3), $mau(bc)$(= 41.5), and $mau(b)$ (= 41). Thus, we can obtain the inequality $mau(bce) \geq mau(bc) \geq mau(b)$. The minimum high average-utility count of each child node is less than or equal to its ancestor. Based on this property, the downward closure of the *auub* model is restored for the case of multiple thresholds, as follows.

Theorem 3 (Sorted downward closure property of *auub*, SDC property)**.** *Let $\prec$ be the processing order, and Y be an extension of itemset X such that $X \subseteq Y$. If Y is a HAUUBI, X is also a HAUUBI.*

Proof. According to Theorem 1 and Property 1, relationships $auub(X) > auub(Y)$ and $mau(Y) \geq mau(X)$ hold. Thus if Y is a HAUUBI ($auub(Y) \geq mau(Y)$), then $auub(X) > auub(Y) \geq mau(Y) > mau(X)$, and X is also a HAUUBI. □

Based on the SDC property, the search space of the set-enumeration tree can be safely reduced while preserving the completeness and correctness for mining HAUIs. Besides, during the search, the currently processed item is only extended with items that succeed it according to the $\prec$ order in the enumeration tree. For example, the item (*c*) will not be extended with the item (*b*) in the enumeration tree. The item (*b*) can be thus removed or ignored when processing the item (*c*) to obtain a lower upper-bound on the utility of the itemset *c* and its extensions. This process can reduce the *tmu* value of a processed transaction and provide a lower upper-bound for each itemset that it contains. Details are given below.

Definition 12 (Irrelevant itemsets in the database)**.** An irrelevant itemset is an itemset that should not be extended with other items according to the processing order $\prec$.

In the running example, the processing oder is $\{b \prec c \prec e \prec d \prec a\}$. Thus item (*b*) can be extended with any of the items succeeding (*b*) such as (*c*), (*e*), (*d*) and (*a*), and item (*c*) can be extended with any of the items succeeding (*c*) such as (*e*), (*d*) and (*a*). The reason for this process is to avoid generating duplicated HAUIs. Thus, irrelevant items can be temporally removed from database or ignored to reduce the transaction-maximum utility (*tmu*). For the sake of convenience, a transaction without its irrelevant items is denoted as T'_q in the following definition.

Definition 13 (Revised transaction-maximum utility, *rmu*)**.** The revised transaction-maximum utility of an itemset X in transaction T'_q is denoted as $rmu(X, T'_q)$, and defined as:

$$rmu(X, T'_q) = max\{u(i_1, T_q), u(i_2, T_q), \ldots, u(i_{|T_q|}, T_q)\}, \quad (13)$$

where $X \prec i_1 \prec i_2 \prec, \ldots, i_{|T_q|}$, $T'_q \subseteq T_q$, and $T_q \setminus X = T'_q$.

The *rmu* is an upper-bound value for the revised transactions. Besides, since irrelevant items suceeding X are temporarily removed to obtain the updated transaction-maximum

utility of the itemset X, it follows that the relationship $rmu(X, T'_q) \leq tmu(X, T_q)$ holds, and ensures the correctness and completeness of the search for HAUIs. Based on this definition, we can further lower the upper-bound value of the itemset X as follows.

Definition 14 (Revised tighter upper-bound model, *rtub*)**.** The revised tighter upper bound of an itemset *X* is denoted as $rtub(X)$, and defined as:

$$rtub(X) = \sum_{X \subseteq T'_q \in D} rmu(X, T'_q). \tag{14}$$

In the running example, consider that all supersets of the itemset (*b*) have been processed. The itemset (*b*) has thus became an irrelevant item and it can be temporally removed from the original database. After that, $rmu(T_2)(= 18) < tmu(T_2)(= 40), rmu(T_3)(= 12) < tmu(T_3)(= 20), rmu(T_6)(= 15) < tmu(T_6)(= 20)$. Therefore, the *rtub* values of itemsets contained in the revised transactions can be greatly reduced. For example, $rtub(ce)$ is calculated as $rtub(ce)$(= 30 + 18 + 15)(= 62), which is less than $auub(ce)$(= 30 + 40 + 15)(= 85) after irrelevant items are temporarily removed. To increase the efficiently after removing irrelevant items, a compact average-utility (CAU)-list structure is designed to avoid performing multiple database scans. For the *rtub* upper-bound, the following theorem is obtained.

Theorem 4. *For an itemset X, the relationship $au(X) \leq rtub(X) \leq auub(X)$ holds.*

Proof. Since $rmu(X, T'_q) \leq tmu(X, T_q)$ holds, $rtub(X) \leq auub(X)$ holds. Also, $au(X) \leq rtub(X)$ since *rtub* is an upper-bound on the average-utility of X. We can then obtain that $au(X) \leq rtub(X) \leq auub(X)$. □

Thus, the proposed *rtub* model is a tighter upper-bound than the *auub* model. The search space can thus be greatly reduced. Since supersets of an itemset are generated by merging the itemset with another itemset sharing a (*k*-1)-itemset as prefix, this theorem will greatly reduce the search space if unpromising supersets can be identified early. An anti-monotonicity property for the *rtub* upper-bound is derived next.

Property 2 (Anti-monotonicity property of *rtub*)**.** *Let an itemset Y be an extension of an itemset X. It follows that $rtub(X) \geq rtub(Y)$.*

Proof. Since $X \subseteq Y$, $tids(Y) \subseteq tids(X)$. Thus, $rtub(X) \geq rtub(Y)$ holds according to the definition of *rtub*. □

The anti-monotonicity property of *rtub* suggests that if the *rtub* value of an itemset is less the minimum high average-utility count, its supersets cannot be HAUIs since $au(Y) \leq rtub(Y) \leq rtub(X)$. Based on this property, a large amount of computation can be avoided and performance of HAUI mining can be improved.

Definition 15 (High revised tighter upper-bound itemset, HRTUBI)**.** An itemset *X* is called a high revised tighter upper-bound itemset (HRTUBI) if its revised tighter upper-bound value is no less than the minimum high average-utility count, defined as:

$$HRTUBI \leftarrow \{X | rtub(X) \geq mau(X)\} \tag{15}$$

A HRTUBI is an itemset that may be an actual HAUI. Based on the *rtub* model, the search space can be reduced since the *rtub* model can provide a lower upper-bound on the average-utilities of itemsets compared to the *auub* model.

Theorem 5 (Sorted downward closure property of *rtub*)**.** *Let $\prec$ be the processing order, and let an itemset Y be an extension of an itemset X such that $X \subseteq Y$. If Y is a HRTUBI, then X is also a HRTUBI.*

Proof. According to Properties 1 and 2, we can obtain that $rtub(X) > rtub(Y)$ and $mau(X) < mau(Y)$ holds. Thus, if *Y* is a HRTUBI ($rtub(Y) \geq mau(Y)$), it follows that $rtub(X) > rtub(Y) \geq mau(Y) > mau(X)$, and X is also a HRTUBI. □

The proposed sorted enumeration tree provides a useful property that guarantees the downward closure property of the traditional *auub* upper-bound and the proposed $rtub$ upper-bound. Thus, all supersets/extensions (child nodes) of an itemset X (parent node) can be directly pruned without missing any HAUIs if $rtub(X)$ or $auub(X)$ are less than $mau(X)$. To efficiently utilize the proposed tighter upper-bound model and its downward closure property, a compact average-utility (CAU)-list structure is developed to keep the information required by the mining process. The CAU-lists of single items can be constructed by performing two database scans. During the first database scan, the *auub* value of each 1-item is calculated. If the *auub* value of a 1-item is no less than the minimum high average-utility count, it is considered as a high average-utility upper-bound 1-item (1-HAUUBI). Items satisfying this condition are then sorted according to the *auub*-ascending order. The items that do not satisfy this condition are removed from the original database. Thus, the revised maximal utility (*rmu*) is obtained. After that, the database is scanned again to construct the CAU-lists of single items. The *rmu* of each transaction T_q is the remaining maximal utility of the itemset *X* in the transaction T_q, which is the transaction maximal utility (*tmu*) among all items except the itemset *X* in the transaction T_q. The definition is given below.

Definition 16 (Compact average-utility (CAU)-list)**.** *The proposed compact average-utility (CAU)-list structure of an itemset X stores relevant information about that itemset and each transaction T_q that contains (X). Each transaction T_q containing X is represented by an entry in the (CAU)-list of X which has three fields: (1) the transaction id of T_q (tid); (2) the utility of (X) in T_q (iutil); (3) the revised maximal utility of (X) in T_q (rmu).*

Note that the proposed CAU-list structure is different from the average-utility (AU)-list structure used by the HAUI-Miner algorithm since the revised maximal utility (*rmu*) of (*X*) in each transaction is stored instead of the transaction-maximum utility (*tmu*). Thus, a tighter upper-bound (*rtub*) for the itemset X can be easily obtained by summing up the *rmu* of each entry

in the list, which can then be used to prune a larger part of the search space compared to the *auub* model. Besides, the CAU-list of each promising item ($auub(i_j) \geq LMAU$) can be easily constructed. Note that each promising itemset should satisfy the condition as: $auub(i_j) \geq LMAU$ since it can guarantee the completeness and correctness of the proposed approach for discovering HAUIs. For the running example, CAU-lists are constructed according to the *mau*-ascending order of items, as shown in Fig. 2.

Sum of iutils ***Sum of rmu***

b	*80*	*80*
tid	***iutils***	***rmu***
2	*40*	*40*
3	*20*	*20*
6	*20*	*20*

c	*84*	*87*
tid	***iutils***	***rmu***
1	*30*	*30*
2	*18*	*18*
3	*12*	*12*
4	*9*	*12*
5	*15*	*15*

e	*45*	*45*
tid	***iutils***	***rmu***
1	*15*	*15*
2	*9*	*9*
5	*6*	*6*
6	*15*	*15*

d	*40*	*42*
tid	***iutils***	***rmu***
2	*8*	*8*
3	*8*	*8*
4	*12*	*12*
6	*12*	*14*

a	*35*	*35*
tid	***iutils***	***rmu***
1	*7*	*7*
3	*7*	*7*
4	*7*	*7*
6	*14*	*14*

Fig. 2: The CAU-list of single items.

The CAU-list structure can be easily used to calculate the average-utility and upper-bound for a *k*-itemset ($k \geq 2$) by applying a very simple join (intersection) operation on the CAU-lists of some of its (*k*-1)-subsets. This is more efficient than using an FP-tree-like structure. The reason is that an FP-tree-based approach needs to recursively build the tree structure (for example, the conditional FP-trees in frequent pattern mining) and mine the required information at each level of these trees, which requires a considerable amount of memory to keep the necessary information to discover the desired itemsets. Using the CAU-list structure, common transactions between two (k-1)-itemsets can be directly identified to generate a *k*-itemset ($k \geq 2$) by comparing the tids in the two corresponding CAU-lists. For example, consider that the lengths of the two CAU-lists are respectively *m* and *n*. Thus, at most (*m* + *n*) comparisons are required for identifying the common transactions since all tids in a CAU-list are ordered. For example, the itemset (*bc*) can be generated by identifying the common tids of the itemsets (*b*) and (*c*). In that case, the number of comparisons to create the CAU-list of *c* is 5, which can be seen in Fig. 3.

B. Proposed algorithm and pruning strategies

In the past, the level-wise HAUIM-MMAU algorithm [29] was designed to discover all HAUIs by considering multiple minimum high average-utility counts. However, it suffers from intrinsic inefficiency since it performs multiple database scans

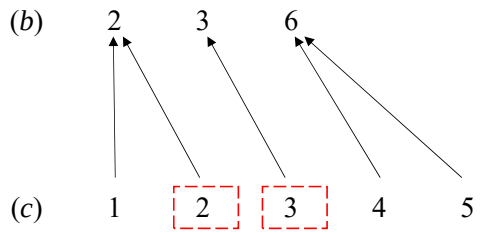

Fig. 3: The comparison of tids for the 2-itemset (*bc*).

and generates an maintains an enormous amount of candidates in memory during the mining phase. To overcome the above drawbacks, this paper introduces a more efficient algorithm for HAUIM with multiple minimum high average-utility counts (MEMU) based on the proposed CAU-list. Besides, to further improve the mining performance by pruning unpromising itemsets early, three pruning strategies are developed. Details are presented next.

According to Theorem 1, for an itemset X, if $auub(X) < LAMU$, the supersets of *X* are not HAUIs. Based on this property, a first pruning strategy is obtained.

Pruning strategy 1. (Remove unpromising items from database, RUI): Each unpromising item i_j, such that $auub(i_j) < LMAU$, is removed from the database to obtain a lower transaction-maximum utility (*tmu*) for each transaction, which can be used to safely reduce the search space while ensuring that all HAUIs can still be discovered.

Proof. Let Y be an extension of the itemset (i_j). According to Theorem 1, it is found that $auub(Y) \leq auub(i_j)$. According to the definition of *LMAU*, it can be concluded that the *LMAU* is the lowest minimum high average-utility count in the database D. Thus, if $auub(i_j) < LMAU$, $auub(Y) \leq auub(i_j) < LMAU < mau(Y) \leq mau(i_j)$. Hence, all extensions of the unpromising itemset are not HAUIs. □

After removing all the unpromising itemsets from transactions, the *tmu* value may be updated to a lower value. Thus, evaluating itemsets based on the *auub* model or *rtub* model can provide lower upper-bound values compared to that of the traditional *auub* model.

The pruning strategy 1 is easily implemented by performing one database scan. For example, the *auub* value of each item in *D* is first calculated by scanning the database once. This allows identifying unpromising items, which are then removed from the database, and the *tmu* value of each transaction is then updated. In previous studies such as HUI-Miner [22], a newly generated itemset (P_{xy}) is obtained by combining two of its subsets P_x and P_y. Thus, if $auub(P_{xy})$ is less than the *LAMU* before constructing its CAU-list, the supersets of P_{xy} can be ignored. To efficiently evaluate *k*-itemsets ($k \geq 2$), the estimated average-utility matrix (EAUM) is developed, presented next.

Definition 17 (Estimated average-utility matrix, EAUM)**.** The EAUM structure is a compact matrix, which contains an

element of the form $< I_x, I_y, auub(I_x \cup I_y) >$ for each pairs of items I_x and I_y that co-occur in the database.

The EAUM for the running example is shown in Table IV. For example, the auub of (ab) is calculated as $auub(ab)$ = $tmu(T_3)$ + $tmu(T_6)$ = 20 + 20 (= 40). The *auub* value of (*ab*) is thus set to 40.

TABLE IV: The EAUM of the running example.

	b	*c*	*d*	*e*
a	40	62	52	50
b	-	60	80	60
c	-	-	72	85
d	-	-	-	60

Pruning strategy 2. (Estimated average-utility matrix strategy, EAUMS): If the *auub* of a 2-itemset according to the EAUM is less than the *LMAU*, all supersets of this itemset are not HAUIs, and can be directly pruned.

Proof. Let *Y* be a superset of an itemset *X* such that $X \subseteq Y$. Based on Theorem 1, $au(X) \leq auub(X)$ and $auub(Y) \leq auub(X) \leq LMAU < mau(Y) < mau(X)$. Thus any supersets of an unpromising 2-itemset can thus be directly pruned and correctness and completeness are preserved for mining HAUIs. □

Hence, the EAUMS can be used to prune unpromising *k*-itemsets ($k \leq 3$) early. Notice that the EAUMS does not preserve the revised tighter upper bound (*rtub*) for 2-itemsets since the remaining maximal utility (*rmu*) of each transaction is kept in each entry.

Recently, the HUP-Miner [15] algorithm was proposed to improve the performance of HUI-Miner [22] for HUIM. Two new pruning strategies, called PU-Prune and LA-Prune, were proposed to further reduce the search space. LA-Prune provides a tighter utility upper-bound value for each itemset used when performing the join operation while generating new itemsets and their utility lists. Here, we further extend the *auub* and *rtub* models to apply the LA-Prune strategy in HAUIM.

Pruning Strategy 3 (LA-Prune strategy): Let P_x and P_y be two itemsets. The join operation does not need to be performed if any of the following equations is less than the minimum high average-utility count when performing the join operation:

$$LAPA(P_x) = auub(P_x) - \sum_{T_q \in tids(P_x) \wedge T_q \notin tids(P_y), T_q \in D} tmu(T_q). \quad (16)$$

$$LAPR(P_x) = rtub(P_x) - \sum_{T_q \in tids(P_x) \wedge T_q \notin tids(P_y), T_q \in D} rmu(T_q). \quad (17)$$

Proof. We have that $auub(P_{xy})$ = $auub(P_x)$ - $\sum_{T_q \in tids(P_x) \wedge T_q \notin tids(P_y)} tmu(T_q) = LAPA(P_x)$, and $rtub(P_{xy})$ =$rtub(P_x)$ - $\sum_{T_q \in tids(P_x) \wedge T_q \notin tids(P_y)} rmu(T_q) = LAPR(P_x)$.
According to **Theorem 1** and **Property 2**, if $LAPA(P_x)$ and/or $LAPR(P_x)$ are less than *LAMU* during the CAU-list construction process, all the supersets of P_x cannot be HAUIs, and the join operation can be stopped. □

This pruning strategy is powerful to improve the performance of HAUIM since it reduces the cost of join operations by stopping some joins early. Hence, based on the proposed upper-bound model and three pruning strategies, the proposed more efficient high average itemset mining algorithm with multiple minimum average utility counts (MEMU) is described in Algorithm 1.

Algorithm 1: MEMU Algorithm

Input: D, a transactional database; $ptable$, a profit table; *MAU-table* = $\{mau(i_1), mau(i_2), \ldots, mau(i_r)\}$, user-specified multiple minimum high average-utility counts.
Output: The set of high average-utility itemsets, HAUIs.

```
// X.AUL, the average-utility list of
   an itemset(X);
// I*, the set of items in D;
// I*.AULs, the set of average-utility
   lists of items in D;
```

1 calculate *auub* of each item;
2 $LMAU \leftarrow min\{mau(i_1), \ldots, mau(i_r)\}$;
3 **if** $auub(i_j) < LMAU$ **then**
4 remove i_j from D;
5 recalculate the *auub* of each remaining item in *D*;
6 sort items in each transaction in *mau*-ascending order;
7 **Search**($\emptyset$,$I^*.AULs$, *EAUM*, *MAU-table*, *LMAU*);
8 return HAUIs;

As shown in algorithm 1, the proposed algorithm first calculates the *auub* of each item by scanning the database once (Line 1) and calculates the least minimum average utility (*LMAU*) using the MAU-Table (Line 2). Pruning strategy 1 is then applied to remove unpromising items from *D* such that each item in I^* is removed if it has a *auub* value that is less than *LMAU* (Line 3). According to Theorem 1 and the previous discussion, it is useful to apply this strategy to prune unpromising items early. After that, the database is scanned again to recalculate the *auub* of each remaining item and construct its CAU-list and the EAUM structure (Line 4). To efficiently discover all HUAIs, all items are then sorted in *mau*-ascending order (Line 5). This process guarantees the downward closure property of the *auub* model and the proposed *rtub* model. After obtaining the CAU-list of each item in I^*, a depth-first **Search** is recursively performed to find all HAUIs. This process is described in Algorithm 2 (Line 6).

As shown in Algorithm 2, the **Search** procedure takes an itemset and a set of 1-extensions of that itemset as arguments. The procedure then sequentially processes each itemset (X_a) in the set of 1-extensions of the itemset (*X*) (Line 1). For each itemset (X_a), the utility of each entry in the CAU-list of X_a is summed up and if that sum is greater than the minimum average utility $mau(X_a)$, then X_a is a high

Algorithm 2: Search Algorithm

Input: *X*, an itemset; *extensionsOfX*, a set of CAU-lists of all 1-items in $I^*.AULs$; *EAUM*, an estimated average-utility matrix; *LMAU*, a least minimum high average-utility count.
Output: The set of high average-utility itemsets, HAUIs.

```
1  for each X_a ∈ extensionsOfX do
2      if sum(X_a.iutils)/|X_a| ≥ mau(X_a) then
3          HAUIs ← HAUIs ∪ X_a;
4      if rtub(X_a) ≥ mau(X_a) then
5          extensionsOfX_a ← φ;
6          for each X_b ∈ extensionsOfX such that
             a ≺ b do
7              if EAUM(X_a, X_b) ≥ LMAU then
8                  X_ab = X_a ∪ X_b;
9                  X_ab ← Construct
                     (X, X_a, X_b, LMAU);
10                 if X_ab.AULs ∉ null then
11                     extensionsOfX_a ←
                         extensionsOfX_a ∪ X_.ab.AUL;
12         Search(X_a, extensionsOfX_a, EAUM,
             MAU-table, LMAU);
13 return HAUIs;
```

Algorithm 3: Construct Algorithm

Input: *X*, an itemset; X_a, the extension of *X* with an item *a*; extensions of *X* with an item *b*; LMAU, the least minimum high average-utility count.
Output: X_{ab}, the CAU-list of X_{ab}.

```
1  set X_ab.AUL ← φ;
2  set rtubOfX_a = rtub(X_a);
3  for each entry E_a ∈ X_a.AUL do
4      if ∃E_b ∈ X_b.AUL ∧ E_a.tids == E_b.tids then
5          if X.AUL ≠ ∅ then
6              if ∃E ∈ X.AUL ∧ E.tid == Ea.tid then
7                  E_ab ←< E_a.tid, E_a.iutil + E_b.iutil −
                     E.iutil, E_a.rmu >;
8          else
9              E_ab ←<
                 E_a.tid, E_a.iutil + E_b.iutil, E_a.rmu >;
10         X_ab.AUL ← X_ab.AUL ∪ E_ab;
11     else
12         rtubOfX_a = rtubOfX_a − E_a.rmu;
13         if rtubOfX_a < LMAU then
14             return ∅;
15 return X_ab.AUL;
```

average-utility itemset (Line 3). After that, the revised tighter upper-bound (*rtub*) of X_a is checked to determine whether the supersets of X_a should be processed by the recursive depth-first search (Line 4). If $rtub(X_a)$ is greater than $mau(X_a)$, the set of CAU-lists of all 1-extensions of X_a is constructed by extending each itemset (X_b) such that $a \prec b$ (Lines 6 to 11). To avoid the costly join operation for unpromising itemsets, each 1-extension of X_a will be checked before constructing its CAU-list. If $EAUM(a, b) \geq LMAU$, X_{ab} is considered as a promising itemset and its CAU-list is thus constructed (Lines 7 to 9). If the returned $X_{ab}.AUL$ is not empty, it should be put into the set $extensionsOfX_a$ for further processing (Lines 10 to 11). After that, the set $extensionsOfX_a$ is further processed to find HAUIs by recursively executing the **Search** algorithm (Line 12). The Construction algorithm for a CAU-list is shown in Algorithm 3.

One of the most critical operation in MEMU is the join operation performed by the **Construct** procedure for constructing the CAU-list of an itemset. The procedure constructs the CAU-list of a new itemset X_{ab} using those of two subsets X_a and X_b that are extensions of an itemset X. At first, the CAU-list of X_{ab} is initialized as empty (Line 1), and the temporary variable $rtubOfX_a$ is set to $rtub(X_a)$ (Line 2). For each entry E_a in $X_a.AUL$, the procedure searches for an entry E_b in $X_b.AUL$ such that $E_a.tid$==$E_b.tid$. If E_b does not exist, the Pruning strategy 3 (*LAPR*) is applied to avoid constructing a large number of entries for an unpromising itemset (Lines 11 to 14). Otherwise, if an entry *E* exists in $X.AUL$ such that $E.tid$==$E_a.tid$, a new entry E_{ab} is built in the CAU-list of X_{ab}. If the CAU-list of *X* is empty, the transaction containing both X_a and X_b does not contain X. Hence, the new entry E_{ab} is thus created as shown in Line 9. Notice that the *rmu* of E_{ab} should be assigned as value for E_a since $a \prec b$ and items in transactions are sorted according to the $\prec$ order. The revised maximal utility (*rmu*) of X_{ab} is the same as that of X_a in the corresponding transactions containing both X_{ab} and X_a.

Based on the proposed upper bound and three pruning strategies, it can be found that the designed MEMU algorithm can efficiently find the whole set of HAUIs while considering multiple minimum average-utility thresholds. The correctness and completeness of the algorithm follows from the previous definitions and proofs.

V. An Illustrated Example

In this section, a detailed example is provided to illustrate the proposed algorithm step-by-step using the database of the running example presented in Tables I and II. In this example, a minimum high average-utility count is assigned to each item, as shown in Table III.

First, the *auub* of each item is calculated by scanning the database once, and unpromising items (having an *auub* value less than the *LMAU*) are identified and removed from the database. For example, item (*f*) in T_6 is removed from the database. After that, the *tmu* value of T_6 is then updated and becomes 20. The *auub* value of each item is then recalculated and updated. The original and updated *auub* values of items are shown in Table V.

Then, all the remaining items are sorted according to the *mau*-ascending order, that is $b \prec c \prec e \prec d \prec a$. This order

TABLE V: The original *auub* value of each 1-itemset.

item	*a*	*b*	*c*	*d*	*e*	*f*
Original *auub*	84	82	102	94	107	22
Updated *auub*	82	80	102	92	105	-

will be used by the depth-first Search procedure to process items in the set ($I^*.AULs$). For example, the item (b) will be first processed and $rtub(b)$ will be calculated as $rtub(b)$(= 40 + 20 + 20)(= 80), which is larger than $mau(b)$(= 41). At the same time, the EAUM is constructed to store information about the relationships between 2-itemsets. The constructed EAUM is shown in Table IV. It is found that all remaining items satisfy the condition and the construction process is thus performed to construct the CAU-lists of all items. To append the item (a) to the item (b) for constructing its 1-extension, the *auub* value of (ba) in the EAUM is then compared with the *LAMU*. In this example, since $auub(ba)$(= 40 < 41), the CAU-list of the 1-extension (ba) is not constructed. However, the CAU-list of (bc) can be constructed since $auub(bc)$(= 60) is no less than the *LMAU*. This process can be done by checking for matching transaction IDs in the CAU-lists of (b) and (c). The CAU-lists of the itemsets (bc)(be)(bd) are respectively constructed. The *LAPR* strategy is then applied to avoid the construction process if there is no entries for 1-extensions. In this example, the itemset (bce) is an unpromising itemset (detected by *LAPR*), since the transaction 3 (tid = 3) is not found in the *tidset* of itemset (be). The *rtub* value of the itemset (bc) is then updated as 40, which is less than *LMAU*(= 41). Thus, itemset (bce) and its supersets are not HAUIs. The depth-first search from itemset (bc) is thus stopped. After that, the depth-first **Search** procedure is recursively applied to find supersets of (be) until no candidates are generated. The same procedure is also applied to other itemsets. The enumeration tree of items (b) and (c), for example, are illustrated in Fig. 4.

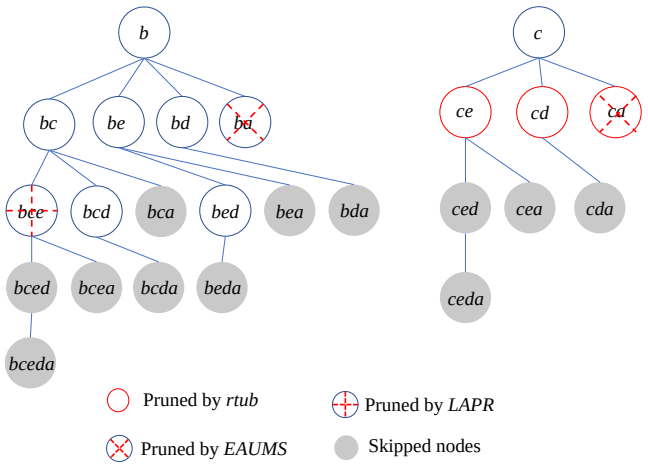

Fig. 4: The enumeration trees of items (b) and (c).

In this example, the efficiency of the proposed *rtub* model and pruning strategies can be observed. For instance, without using the *rtub* model, itemsets (ce) and (cd), and their supersets cannot be pruned early using the traditional *auub* model. Besides, the EAUM plays a significant role to prune unpromising itemsets such as itemset (ba) and its supersets, which can be directly pruned before constructing their CAU-lists. This process is then repeatedly performed until no candidates are generated. Then, the final set of HAUIs when considering multiple thresholds has been obtained. This set contains three itemsets: {b:80, c:84 bc:41.5}.

VI. Experimental Evaluation

In this section, the performance of the proposed MEMU algorithm is compared to the state-of-art HAUIM-MMAU algorithm for mining the HAUIs while considering multiple minimum high average-utility counts. Moreover, to evaluate the efficiency of the pruning strategies, two versions of the MEMU algorithm are compared. The version without pruning strategies is named MEMU-, while the version with all three pruning strategies is called MEMU+. All algorithms in the experiments are implemented in Java and experiments are performed on a computer equipped with an Intel(R) Core(TM) i3-2350 2.3GHz processor and 6 GB of main memory, running the 64 bit Microsoft Windows 10 operating system. Experiments were conducted on five real-world datasets [9] and one synthetic [14] dataset. A simulation model [18] was developed to generate the purchase quantities and unit profits of items in transactions for all datasets. A log-normal distribution was used to randomly assign purchase quantities in the [1, 5] interval and item profits values in the [1, 1000] interval. Parameters descriptions and the characteristics of the six datasets are shown in Tables VI and VII.

TABLE VI: Parameters of the datasets.

#|**D**|	Total number of transactions
#|**I**|	Number of distinct items
AveLen	Average length of transactions
MaxLen	Maximal length of transactions
Type	Database type (sparse or dense)

TABLE VII: Characteristics of the datasets.

Dataset	#|**D**|	#|**I**|	**AvgLen**	**MaxLen**	**Type**
retail	88,162	16,407	10	76	Sparse
kosarak	990,002	41,270	8	2,498	Sparse
accidents	340,183	469	51	33.8	Sparse
T40I10D100K	100,000	1,000	39.6	77	Sparse
chess	3,196	76	37	37	Dense
mushroom	8,124	120	23	23	Dense

To generate the *mau* value of each item for the proposed algorithm, we have been inspired by the solution proposed in [10] to assign multiple minimum thresholds to items in HUIM. The following equation is used to automatically set the *mau* value of each item:

$$mau(i_j) = max(\beta \times p(i_j), glmau), \tag{18}$$

where β is a constant value, which is used to multiply the profit of the item $p(i_j)$. The global least minimum average utility is a user-defined parameter denoted as *glmau*. When β is set to zero, *glmau* is the same as the single minimum high average-utility count. In this case, the problem of HAUIM with multiple minimum high average-utility threshold is equal to the

traditional problem of HAUIM with a single minimum high average-utility threshold. To ensure randomness, β is randomly selected from a user-defined interval, for example, the interval [100, 1000] is used for the retail and T10I4D100K datasets; the interval [7000, 70000] is used for the kosarak dataset and the interval [1000, 10000] is used for the other datasets.

A. Runtime

In this section, we first compare the runtime of all algorithms to evaluate their efficiency for various *glmau* and a fixed interval from which β is randomly selected. The runtimes include both the processing time for the mining phase and the I/O cost. Results are shown in Fig. 5 for a fixed randomly generated β and various *glmau* values.

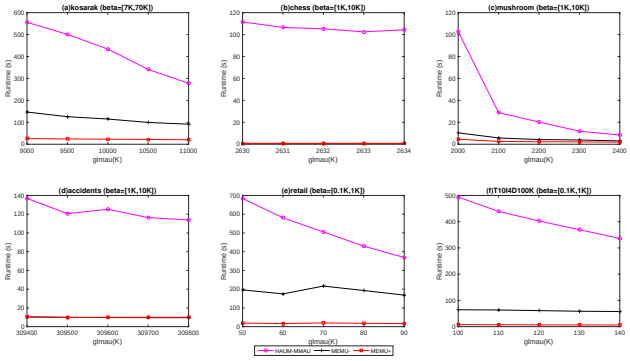

Fig. 5: Runtimes for a fixed β and various *glmau* values.

In Fig. 5, it can be observed that the proposed algorithm outperforms the state-of-the-art HAUIM-MMAU algorithm and that the proposed *rtub* model is more effective at reducing the search space than the traditional *auub* model. Using the proposed *rtub* model and three pruning strategies, the MEMU+ algorithm is generally up to one or two orders of magnitude faster than the HAUIM-MMAU algorithm. For example in Fig. 5(a), when *glmau* is set to 9000K, the runtime of HAUIM-MMAU for discovering the complete of HAUIs requires 500 seconds while the MEMU+ algorithm needs less than 24 seconds. The reason is that HAUIM-MMAU is an Apriori-like approach, which performs multiple database scans to find candidates at each level (itemsets of each size), and must keep many of them in memory during the search. The proposed algorithm utilizes a list structure to represent the database vertically and explore promising itemsets in the enumeration tree, which is more efficient than the level-wise approach. Thus, MEMU- outperforms HAUIM-MMAU, as it can be observed in Fig. 5(a) to 5(f). Besides, the three pruning strategies play an essential role in enhancing the mining performance of the proposed algorithm since unpromising itemsets can be directly ignored in the enumeration tree. The runtimes for a fixed *glmau* and various β values are shown in Fig. 6.

It can be observed in Fig. 6 that the proposed *rtub* model used by the proposed algorithm is more effective than the traditional *auub* model used by the state-of-the-art HAUIM-MMAU algorithm. The reason is the same as the one for explaining the results of Fig. 5. Moreover, it is observed that

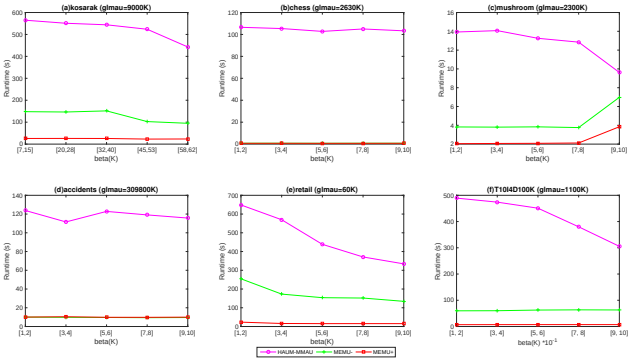

Fig. 6: Runtimes for a fixed *glmau* and various β values.

the runtime decreases as β is increased since the *mau* of each item becomes larger, and the number of HAUIs decreases as well as the part of the search space that is explored. In summary, the proposed MEMU algorithm with the *rtub* model (with or without pruning strategies) is more efficient than the HAUIM-MMAU algorithm based on the traditional *auub* model.

B. Memory Usage

This section compares the memory usage of the algorithms. The memory usage is measured using the standard Java API and the peak memory usage of each compared algorithm is considered as its final memory usage for each dataset. The results for the six datasets and various *glmau* values and a fixed β are shown in Fig. 7.

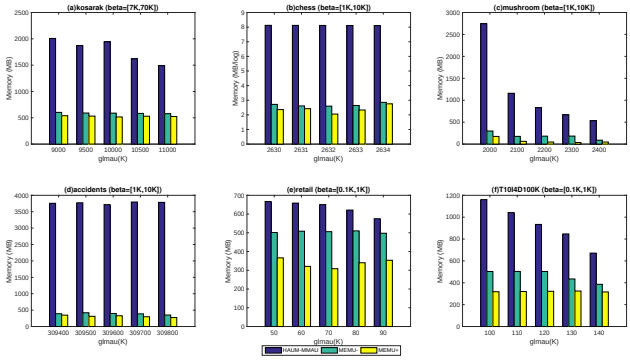

Fig. 7: Memory usage for a fixed β value and various *glmau* values.

In Fig. 7, it can be observed that the proposed MEMU+ algorithm is more memory-efficient than the state-of-the-art HAUIM-MMAU algorithm. The *rtub* model can efficiently reduce the memory usage of the algorithm since the number of nodes that are explored in the enumeration tree by MEMU+ is much smaller than that of the HAUIM-MMAU algorithm based on the *auub* model. Thus, multiple database scans are avoided and the proposed algorithm do not need to maintain a large number of promising itemsets in memory during the mining phase. For example in Fig. 7(a), the HAUIM-MMAU algorithm requires 3,370 MB of memory while the MEMU-

and MEMU+ respectively uses 602 MB and 539 MB of memory when *glmau* is set to 9000K.

Thus, it can be found that the proposed approach can reduce the memory usage by up to 5 times compared to the traditional model. The results on six datasets for various β values and a fixed *glmau* are shown in Fig. 8.

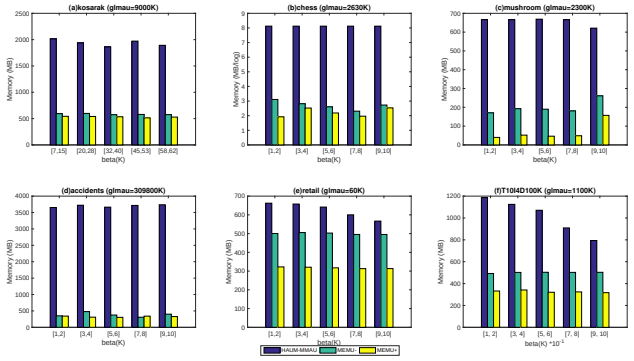

Fig. 8: Memory usage for a fixed *glmau* and various β values.

In Fig. 8, it can also be observed that the proposed algorithms outperform the generate-and-test HAUIM-MMAU algorithm for various β values. The reason is the same as for Fig. 7. In summary, the proposed *rtub* model and pruning strategies allows to reduce the search more effectively and the algorithm is designed to keep less information in memory during the mining process.

C. Number of Candidates

In this section, the number of candidates for all algorithms is compared for various *glmau* values and a fixed β. Results for the six datasets are shown in Fig. 9.

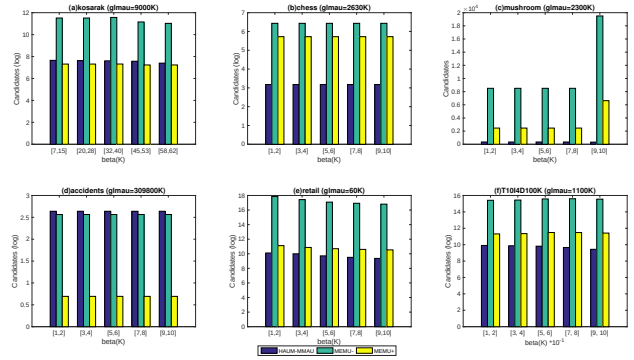

Fig. 9: Number of candidates for a fixed β and various *glmau* values.

In Fig. 9, it can be observed that the number of candidates generated by MEMU on the six datasets is not always less than that of HAUIM-MMAU. For example in Fig. 9(a) and 9(b), the number of candidates for MEMU is much less than that of HAUIM-MMAU. The result is acceptable since HAUIM-MMAU uses two pruning strategies to reduce the number of generated candidates. However, those two pruning strategies are not time and memory efficient, as it can be seen in the above results. Besides, the HAUIM-MMAU algorithm suffers from the intrinsic weakness that multiple database scans are performed and that a very large number of candidates must be kept in memory.

In addition, it is also observed that the proposed *rtub* model is more effective than the traditional *auub* model since one to two orders of magnitude less candidates are produced, as shown in Fig. 9. For example, in Fig. 9(d), HAUIM-MMAU generates 1,057 candidates while MEMU generates 13 candidates when the *glmau* is set to 309,400K. From this result, it can be concluded that the influence of the *rtub* model is related to the characteristics or distribution of the datasets. The effectiveness of the *rtub* model is influenced by the *mau*-ascending order of items in transactions. The HAUIM-MMAU is based on *auub* model to reduce search space. Although it generates less candidates in some specified cases, for example, in Figs. 9(b), 9(e), and 9(f); it takes, however, more computations in terms of runtime and memory usage to find the required information. The reasons are that the compared HAUIM-MMAU algorithm adopts the *auub* model with their pruning strategies; more candidates are reduced but it takes long time for exploring the search space and needs extra memory to handle the promising candidates. In the designed algorithms, the three proposed pruning strategies have great effects on the pruning of unpromising candidates. For example in Fig. 9(e), when the *glmau* is set to 50K, MEMU- generates 50,470,293 candidates but MEMU+ only generates 80,926 candidates. The results on six datasets for various β values and a fixed *glmau* are shown in Fig. 10 .

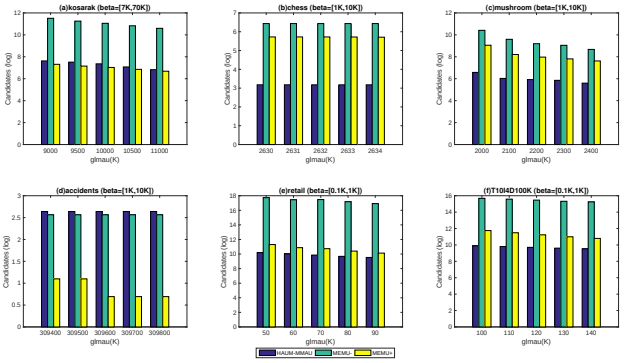

Fig. 10: Number of candidates for a fixed *glmau* and various β values.

In Fig. 10, it can be clearly observed that the compared algorithms have similar results as shown in Fig. 9. The same reasons can be used to explain the result. In summary, the proposed *rtub* model and three pruning strategies possess great pruning ability to eliminate candidate nodes in the enumeration tree.

D. Scalability

Since MEMU+ extends MEMU- with several optimizations, MEMU+ is the final proposed algorithm. Thus the scalability of the MEMU+ algorithm for various dataset

sizes T10I4N4KD|X|K is compared with the state-of-the-art HAUIM-MMAU algorithm in terms of runtime, memory usage and number of candidates. The variable X, indicating the dataset size, is varied from 100(K) to 500(K) transactions using an increments of 100(K) transactions. The *glmau* is set to 200K and β is randomly selected in the [1K, 10K] interval. Results are shown in Fig. 11.

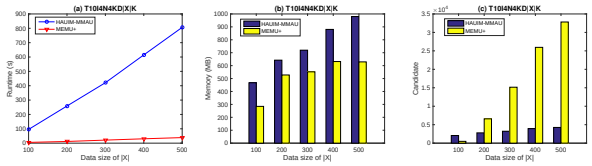

Fig. 11: Scalability for various dataset sizes.

In Fig. 11, it can be clearly observed that the proposed algorithm has better scalability than the state-of-the-art HAUIM-MMAU algorithm as dataset size is increased, in terms of runtime and memory usage. As shown in Fig. 11(a), the execution time of HAUIM-MMAU rapidly increases while that of MEMU remains stable. Besides, it can be observed that the memory usage is considerably less for MEMU compared to HAUIM-MMAU. For example, HAUIM-MMAU requires 467M of memory while MEMU+ needs 284M when X is set to 100K. The effectiveness and efficiency of MEMU+ can thus be observed. However, the number of candidates generated by MEMU+ is still larger than that of HAUIM-MMAU. This is because HAUIM-MMAU employs two pruning strategies that greatly reduce the search space but has to exhaustively search all sub-itemsets of the current processed itemset to check the condition for pruning unpromising itemsets. Thus, although the number of candidates can be pruned greatly, the pruning operation is inefficient. Based on the above discussion, these results are reasonable and acceptable. Besides, HAUIM-MMAU suffers from the intrinsic weakness of generate-and-test approaches that it must perform multiple database scans. It thus requires a large amount of time and memory to discover patterns. On the contrary, the proposed algorithm employs the compact CAU-list structure to avoid repeatedly scanning the database and the information is efficiently stored in CAU-lists by the mining process. Thus, although the number of candidates for HAUIM-MMAU is less than that of MEMU+, MEMU+ is still more efficient than HAUIM-MMAU in terms of runtime and memory usage. In summary, it can be concluded that the proposed algorithm outperforms the state-of-the-art HAUIM-MMAU algorithm in terms of scalability.

E. Efficiency of the Pruning Strategies

In this section, the effectiveness of the three developed pruning strategies is evaluated on a synthetic dataset for various *glmau* values and a fixed β. The version of the algorithm without any pruning strategies is selected as baseline algorithm, and is denoted as N1. The version of the algorithm that only applies the RUI strategy is denoted as N2. A version of the algorithm that applies both the RUI and LAPR strategies is denoted as N3. Finally, N4 denotes a version of the algorithm that applies all three pruning strategies. The results in terms of runtime, memory usage and number of candidates are shown in Fig. 12.

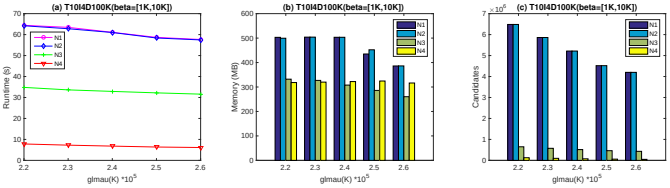

Fig. 12: Efficiency of the developed pruning strategies.

In Fig. 12, it can be observed that each pruning strategy is useful to reduce the search space and speed up the mining process using the designed algorithm. For example, in Fig. 12(a) and (c), when the LAPR strategy is applied in the designed algorithm, N3 requires 34 second and considers 99,970 candidates to complete the mining task, while N2 spends 65 second and considers 18,866 candidates when *glmau* is set to 2.2×10^5. This is because LAPR is a powerful strategy for identifying unpromising candidate nodes early during the construction process of CAU-list structures. Notice that when the *glmau* is greater than 2.4×10^5, as shown in Fig. 12(b), the memory usage of N2 is slightly greater than that of N3. This is because the *EAUMS* strategy utilizes a compact matrix to detect unpromising candidates, which consumes at most $O(n^2)$ memory. When the reduced search space by *EAUMS* is less than its consumed memory, the above situation happens. However, the *EAUMS* has greater pruning ability when the *glmau* is set to smaller values. Thus, we may conclude that the EAUMS strategy is efficient to reduce the memory usage although it may require some additional memory to keep the required information.

F. Complexity Analysis

The complexity of the proposed algorithm is analyzed as follows. If there are m items in a database, in the worst case, all $2^m - 1$ possible itemsets that can be generated using these items are promising and must be explored by the algorithm. For each itemset, a CAU-list is built in the worst case. Constructing a CAU-list for single items is done in log-linear time by scanning the database, while constructing the CAU-list of larger itemsets is done in linear time by comparing two or three CAU-lists of subsets using a two-way or three-way comparison. In the worst case, the CAU-list of an itemset contains an entry for each transaction in the database, which is linear with respect to the database size. Applying the pruning strategies is also done in linear time and space, except for the EAUMS strategy which requires to build a $m \times m$ matrix (exponential size). But this matrix is only built once, and could be deactivated or implemented as a sparse matrix if required.

For the complexity analytics, the number of candidates in the search space is denoted as C. Based on the provided theorems, the developed upper-bound model is tighter than the traditional *auub* model. The best case is that C is equal to m; however, C is usually much greater than m in all realistic cases. Empirically, the gap between C and m becomes huge when the minimum high average-utility threshold is set lower

but it should have different performance in different datasets. Assume that the size of dataset is set as n, each new itemset can be generated with $2 \times n$ (for two way comparisons) or $2 \times n \times log(n)$ (for three way comparisons) times by performing the intersection operation of two CAU-list. For the EAUMS strategy, at most $n \times m^2$ time and m^2 memory are needed to construct the matrix in the first candidate scan to retrieve the related elements. At first, the database is needed to be scanned once and the time complexity is $n \times m$. Thus, the worst case of the time complexity for the developed algorithm is $O(n \times m + 2 \times n \times \times log(n) \times C + n \times m^2)$. The CAU-list requires at most $3 \times n$ memory in the list structure to keep the necessary candidates. Therefore, the worst case of memory usage will be $O(m^2 + 3 \times C \times n)$, and in the worst case, C is equal to 2^m. However, the worst case is not usually happened unless the threshold is set as 0.

Generally, the complexity of the algorithm can be deemed as pseudo-polynomial, as the complexity is roughly linear in time and space for each pattern considered in the search space, while the number of patterns in the search space depends on the effectiveness of the pruning strategies for each dataset.

VII. Conclusion and Future Work

This paper has presented an efficient algorithm called MEMU to discover high average-utility itemsets with multiple minimum high average-utility counts. A novel MAU-list structure and a sorted enumeration (SE)-tree have been developed to mine HAUIs while reducing the search space. A tighter upper-bound model named *rtub* was also designed to prune unpromising itemsets using low upper-bound values compared to the traditional *auub* model. Three pruning strategies based on the *rtub* model were proposed to reduce the search space for mining HAUIs. A series of experiments have been conducted to compare the performance of the proposed MEMU algorithm with the state-of-the-art HAUIM-MMAU algorithm on several datasets in terms of effectiveness, efficiency and scalability.

Traditional HUIM and HAUIM only consider positive unit profits for items. In real-life situations, especially for the analysis of customer transactions, items sometimes have negative weights or unit profits. Some studies have proposed solutions to adapt HUIM to consider negative unit profits. In HAUIM, no studies have yet considered items with negative unit profits. This will be considered in our future work.

Acknowledgment

This research was partially supported by the National Natural Science Foundation of China (NSFC) under grant No. 61503092, by the Shenzhen Technical Project under JCYJ20170307151733005, by the Science Research Project of Guangdong Province under grant No. 2017A020220011, and by the National Science Funding of Guangdong Province under Grant No. 2016A030313659.

References

[1] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," *International Conference on Very Large Data Bases*, pp. 487–499, 1994.

[2] R. Agrawal and R. Srikant, "Mining sequential patterns," *International Conference on Data Engineering*, pp. 3–14, 1995.

[3] D. Arthur and S. Vassilvitskii, "K-means++: the advantages of careful seeding," *Eighteenth Acm-Siam Symposium on Discrete Algorithms. Society for Industrial and Applied Mathematics*, pp. 1027–1035, 2007.

[4] C. F. Ahmed, S. K. Tanbeer, B. S. Jeong, and Y. K. Lee, "Efficient tree structures for high utility pattern mining in incremental databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 12, pp. 1708–1721, 2009.

[5] C. J. Chu, V. S. Tseng, and T. Liang, "An efficient algorithm for mining high utility itemsets with negative item values in large databases," *Applied Mathematics and Computation*, vol. 215, no. 2, pp. 767–778, 2009.

[6] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," *ACM SIGKDD International Conference*, pp. 785–794, 2016.

[7] A. Erwin, R. P. Gopalan, and N. R. Achuthan, "Efficient mining of high utility itemsets from large datasets," *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 554–561, 2008.

[8] J. C. W. Lin, P. Fournier-Viger, and W. Gan "FHN: Efficient mining of high-utility itemsets with negative unit profits," *Knowledge-Based Systems*, vol. 111, no. 1, pp. 283–298, 2016.

[9] P. Fournier-Viger, J. C. W. Lin, A. Gomariz, T. Gueniche, A. Soltani, Z. Deng, and H. T. Lam, "The SPMF open-source data mining library version 2 and beyond," *The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery*, pp. 36–40, 2016.

[10] W. Gan, J. C. W. Lin, and P. Fournier-Viger, "More efficient algorithms for mining high-utility itemsets with multiple minimum utility thresholds," *Database and Expert Systems Applications*, pp. 202–213, 2016.

[11] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining frequent patterns without candidate generation: a frequent-pattern tree approach," *Data Mining and Knowledge Discovery*, vol. 8, no. 1, pp. 53–87, 2004.

[12] Y. Hu and Y. Chen, "Mining association rules with multiple minimum supports: a new mining algorithm and a support tuning mechanism," *Decision Support Systems*, pp. 1–24, 2006.

[13] T. P. Hong, C. H. Lee, and S. L. Wang, "Effective utility mining with the measure of average utility," *Expert System with Applacation*, vol. 38, no. 7, pp. 8259–8265, 2011.

[14] IBM Quest Data Mining Project, Quest Synthetic Data Generation Code, http://www.almaden.ibm.com/cs/quest/syndata.html, 1996.

[15] S. Krishnamoorthy "Pruning strategies for mining high utility itemsets," *Expert Systems with Applications*, vol. 42, no. 5, pp. 2371–2381, 2015.

[16] D. Kim and U. Yun, "Efficient mining of high utility pattern with considering of rarity and length," *Applied Intelligence*, vol. 45(1), pp. 152–173, 2016.

[17] B. Liu, W. Hsu, and Y. Ma, "Mining association

rules with multiple minimum supports," *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1–5, 1999.

[18] Y. Liu, W. Liao, and A. Choudhary, "A fast high utility itemsets mining algorithm," *International Workshop on Utility-based Data Mining*, pp. 90–99, 2005.

[19] C. Lucchese, S. Orlando, and R. Perego, "Fast and memory efficient mining of frequent closed itemsets," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 1, pp. 21–36, 2006.

[20] Y. C. Li, J. S. Yeh, and C. C. Chang, "Isolated items discarding strategy for discovering high utility itemsets," *Data and Knowledge Engineering*, vol. 64, no. 1, pp. 198–217, 2008.

[21] C. W. Lin, T. P. Hong, and W. H. Lu, "Efficiently mining high average utility itemsets with a tree structure," *The International Conference on Intelligent Information and Database Systems*, pp. 131-139, 2010.

[22] M. Liu and J. Qu, "Mining high utility itemsets without candidate generation," *The International Conference on Information and Knowledge Management*, pp. 55–64, 2012.

[23] G. C. Lan, T. P. Hong, and V. S. Tseng, "Efficiently mining high average-utility itemsets with an improved upper-bound strategy," *International Journal of Information Technology & Decision Making*, vol. 11, no. 5, pp. 1009-1030, 2012.

[24] T. Lu, B. Vo, H. T. Nguyen, and T. P. Hong, "A new method for mining high average utility itemsets," *Computer Information Systems and Industrial Management*, pp. 33–42, 2014.

[25] G. C. Lan, T. P. Hong, J. P. Huang, and V. S. Tseng, "On-shelf utility mining with negative item values", *Expert Systems with Applications*, vol. 41, pp. 3450–3459, 2014.

[26] J. C. W. Lin, W. Gan, P. Fournier-Viger, T. P. Hong, and V. S. Tseng, "Fast algorithms for mining high-utility itemsets with various discount strategies," *Advanced Engineering informatics*, vol. 30(2), pp. 109–126, 2016.

[27] J. C. W. Lin, W. Gan, P. Fournier-Viger, T. P. Hong, and J. Zhan, "Efficient mining of high-utility itemsets using multiple minimum utility thresholds," *Knowledge-Based Systems*, vol. 113, pp. 100–115, 2016.

[28] J. C. W. Lin, T. Li, P. Fournier-Viger, T. P. Hong, J. Zhan, and M. Voznak, "An efficient algorithm to mine high average-utility itemsets," *Advanced Engineering Informatics*, vol. 30, no. 2, pp. 233–243, 2016.

[29] J. C. W. Lin, T. Li, P. Fournier-Viger, T. P. Hong, and J. H. Su "Efficient mining of high average-utility itemsets with multiple minimum thresholds," *Industrial Conference on Data Mining*, pp. 14–28, 2016.

[30] J. Liu, K. Wang, and B. C. M. Fung, "Mining high utility patterns in one phase without generating candidates," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 5, pp. 1245–1257, 2016.

[31] J. C. W. Lin, W. Gan, P. Fournier-Viger, T. P. Hong, and T. P. Hong, "FDHUP: Fast algorithm for mining discriminative high utility patterns," *Knowledge and Information Systems*, vol. 51, vol. 3, pp 873–909, 2017.

[32] J. C. W. Lin, S. Ren, and P. Fournier-Viger, "EHAUPM: Efficient high average-utility pattern mining with tighter upper bounds," IEEE Access, vol. 5, pp. 12927–12940, 2017.

[33] J. C. W. Lin, W. Gan, P. Fournier-Viger, T. P. Hong, and V. S. Tseng, "Efficiently mining uncertain high-utility itemsets," Soft Computing, vol. 21 no. 11, pp 2801–2820, 2017.

[34] J. C. W. Lin, S. Ren, P. Fournier-Viger, T. P. Hong, J. H. Su, and B. Vo, "A fast algorithm for mining high average-utility itemsets," Applied Intelligence, pp. 1–16, 2017.

[35] J. Pei, J. Han, B. Mortazavi-SaI, J. Wang, H. Pinto, Q. Chen, and M. C. Hsu, "Mining sequential patterns by pattern-growth:the prefixSpan approach," *IEEE Transactions of Knowledge and Data Engineering*, vol. 16, no. 11, pp. 1424–1440, 2004.

[36] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Fast and memory efficient mining of frequent closed itemsets," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 1, pp. 21–36, 2006.

[37] H. Ryang and U. Yun, "High utility pattern mining over data streams with sliding window technique," *Expert Systems with Applications*, vol. 57, pp. 214–231, 2016.

[38] H. Ryang, U. Yun, and K. Ryu, "Fast algorithm for high utility pattern mining with the sum of item quantities," *Intelligent Data Analysis*, vol. 20, no. 2, pp. 395–415, 2016.

[39] B. E. Shie, V. S. Tseng, and P. S. Yu, "Online mining of temporal maximal utility itemsets from data streams," *ACM Symposium on Applied Computing*, pp. 1622–1626, 2010.

[40] I. W. Tsang, J. T. Kwok, and P. M. Cheung, "Core Vector Machines: Fast SVM training on very large data sets," *Journal of Machine Learning Research*, pp. 363–392, 2005.

[41] V. S. Tseng, B. Shie, C. Wu, and P. S. Yu, "Efficient algorithms for mining high utility itemsets from transactional databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 8, pp. 1772–1786, 2013.

[42] V. S. Tseng, C. W. Wu, P. Fournier-Viger, and P. S. Yu, "Efficient algorithms for mining the concise and lossless representation of high utility itemsets," *IEEE Transactions of Knowledge and Data Engineering*, vol. 27, no. 3, pp. 726–739, 2015.

[43] T. Uno, M. Kiyomi, and H. Arimura, "Efficient mining algorithms for frequent/closed/maximal itemsets," *IEEE ICDM Workshop on Frequent Itemset Mining Implementations*, pp.1–11, 2004.

[44] H. Yao, H. J. Hamilton, and C. J. Butz, "A foundational approach to mining itemset utilities from databases," *SIAM International Conference on Data Mining*, pp. 482–486, 2004.

[45] U. Yun, H. Ryang, G. Lee, and H. Fujita, "An efficient algorithm for mining high utility patterns from incremental databases with one database scan," *Knowledge-Based Systems*, vol. 124, no. 15, pp. 188–206, 2017.

[46] M. J. Zaki, "Scalable algorithms for association mining," *IEEE Transactions on Knowledge and Data Engineering*,

vol. 12, pp. 372–390, 2000.

[47] M. J. Zaki and K. Gouda, "Fast vertical mining using diffsets," *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 362–335, 2003.