EHAUPM: Efficient High Average-Utility Pattern Mining with Tighter Upper-Bounds

Jerry Chun-Wei Lin[1], *Member, IEEE*, Shifeng Ren[1], Philippe Fournier-Viger[2]
Tzung-Pei Hong[3,4], *Member, IEEE*
[1]School of Computer Science and Technology, Harbin Institute of Technology Shenzhen Graduate School, Shenzhen, China
[2]School of Humanities and Social Sciences, Harbin Institute of Technology Shenzhen Graduate School, Shenzhen, China
[3]Department of Computer Science and Information Engineering, National University of Kaohsiung, Kaohsiung, Taiwan
[4]Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, Taiwan

High-utility itemset mining (HUIM) has become a popular data mining task, as it can reveal patterns having a high-utility, contrarily to frequent pattern mining (FIM), which focuses on discovering frequent patterns. High average-utility itemset mining (HAUIM) is a variation of HUIM that provides an alternative measure, called the average utility, to select patterns by considering both their utilities and lengths. In the last decades, several algorithms have been developed to mine high average-utility itemsets (HAUIs). But most of them consume large amounts of memory and have long execution times since they generally utilize the average-utility upper-bound (*auub*) model to overestimate the average-utilities of itemsets. To improve the performance of HAUIM, this paper proposes two novel tighter upper-bound models as alternative to the traditional *auub* model for mining HAUIs. The looser upper-bound model (*lub*) considers the remaining-maximum utility in transactions to reduce the upper-bound on the utilities of itemsets. The second upper-bound (*rtub*) model ignores irrelevant items in transactions to further tighten the upper-bound. Three pruning strategies are also designed to reduce the search space for mining HAUIs by a greater amount, compared to the state-of-the-art HAUI-Miner algorithm. Experiments conducted on several benchmark datasets show that the designed algorithm integrating the two novel upper-bound models outperforms the traditional HAUI-Miner algorithm in terms of runtime, memory usage, number of join operations, and scalability.

Index Terms—**High average-utility pattern, tighter upper-bounds, utility mining, pruning strategy, data mining.**

I. Introduction

The main purpose of data mining techniques is to reveal important, interesting and potentially useful information in large databases [1, 2, 9]. Frequent itemset mining (FIM) [1] plays a key role in data mining since it can reveal frequently purchased sets of items (itemsets) in transactional databases, which is a fundamental research issue in data mining, having multiple real-life applications [5, 10, 30, 34]. Traditional FIM only takes occurrence frequencies of itemsets into account, and does not consider other factors that can help to evaluate the importance of itemsets such as purchase quantities, unit profits of items, and generally the interestingness or weights of items. As a result, the information extracted by traditional FIM algorithms is insufficient for many applications.

To reveal more useful and meaningful information from transactional databases, the task of high utility itemset mining (HUIM) was proposed by Yao et al. [32]. It considers both purchase quantities and unit profits of items, to find the set of high-utility itemsets (HUIs). In HUIM, item quantities in a database are called the internal utilities of items, and the item unit profits are called the external utilities. A minimum utility threshold must be set by the user to find HUIs. The task of HUIM can be viewed as an extension of FIM. To discover HUIs efficiently by reducing the search space of itemsets, the transaction-weighted utility (TWU) model [16] was designed. It provides a downward closure (DC) property for HUIM called the transaction-weighted downward closure (TWDC) property. This property allows to only consider a small set of candidate itemsets called the high transaction-weighted utilization itemsets (HTWUIs) as potential HUIs. Several algorithms [3, 18, 23, 31] were proposed to mine HUIs using the TWU model and additional improvements have been proposed to speed up their performance [21, 26].

Although HUIM can reveal more useful information compared to traditional FIM, it suffers from an important problem. It is that the length of each itemset is not considered when measuring its utility. This is unfair as the utilities of itemsets tend to be greater for itemsets containing more items. To address this issue and provide a more fair measurement of the utilities of itemsets for real-life applications, high average-utility itemset mining (HAUIM) [13] was proposed. It relies on the average-utility measure, which divides the utility of an itemset by its length (number of items that it contains). The goal of HAUIM is to find the set of all high average-utility itemsets (HAUIs). Hong et al. [13] first designed a two-phase TPAU algorithm to mine HAUIs using an Apriori-like approach. The average-utility upper-bound (*auub*) model was designed to ensure the completeness and correctness of the algorithm for mining HAUIs. To speed up HAUIM, a projection-based PAI algorithm [22], a tree-based high average-utility pattern HAUP-tree algorithm [20], and a HAUI-tree [25] algorithm were designed to efficiently mine HAUIs, based on the TPAU algorithm. The HAUI-Miner algorithm [27] was then developed to further enhance the mining performance using a designed average-utility (AU)-list structure. This algorithm is currently the state-of-the-art HAUIM algorithm. Nonetheless, HAUI-Miner suffers from the problem of performing a costly join operation numerous times for mining HAUIs since it

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

adopts the *auub* model to overestimate the utility of itemsets. Because the $auub$ model provides a loose upper bound on the utilities of itemsets, unpromising itemsets with low relative average-utilities cannot be pruned early from the search space. In this paper, an efficient algorithm for high average-utility pattern mining (EHAUPM) is presented for mining HAUIs and a modified average-utility (MAU)-list structure is developed to keep additional information to facilitate mining HAUIs. Two tighter upper-bounds are presented to prune unpromising itemsets early, and thus reduce the search space for mining HAUIs. Moreover, three efficient pruning strategies are designed to speed up the mining performance. The major contributions of this paper are summarized as follows.

1) Two tighter upper-bounds are proposed to greatly reduce the search space for mining HAUIs. The looser upper-bound (*lub*) model is first designed to consider the average-utilities of itemsets and their remaining-maximum utilities in transactions. The second revised tighter upper-bound (*rtub*) model is further designed to ignore irrelevant itemsets in transactions, which can be used to further reduce the search space for mining HAUIs.
2) A modified average-utility (MAU)-list structure is developed to reduce the number of database scans and store the required information for mining HAUIs. Three pruning strategies are also developed to enhance the performance of HAUI mining using the two designed models and MAU-list structure.
3) Extensive experiments are conducted on both real-world and synthetic datasets to show that the proposed algorithm significantly outperforms the state-of-the-art HAUI-Miner algorithm in terms of runtime, memory usage, number of join operations and scalability.

The rest of the paper is organized as follows. A literature review of HUIM and HAUIM is presented in Section II. The preliminaries and problem statement are provided in Section III. The two proposed upper-bounds, the designed EHAUPM algorithm and the three novel pruning strategies are presented in Section IV. An illustrated example describing how the designed algorithm is applied step-by-step is given in Section V. Results of extensive experiments on several benchmark datasets are reported in Section VI. Lastly, a conclusion and a brief discussion of opportunities for future work are presented in Section VII.

II. Literature Review

Association rule mining (ARM) and frequent itemset mining (FIM) are fundamental data mining tasks [1]. They have been widely studied as they can reveal itemsets having high occurrence frequencies in databases, which has many applications. Apriori [1] is the first algorithm for mining association rules (ARs). It performs two phases. In the first phase, it extracts the set of frequent itemsets (FIs) respecting a user-specified minimum support threshold. Then, the derived FIs are combined to form the set of ARs, and only those respecting a user-specified minimum confidence threshold are shown to the user. The Apriori algorithm is said to adopt a generate-and-test approach, since it produces candidates and perform multiple database scans to derive the actual ARs from these candidates. To speed up the discovery of FIs, a compact tree structure called frequent pattern (FP)-tree [12] was developed and a corresponding mining algorithm called FP-growth was designed to derive the set of FIs from the FP-tree structure. Besides Apriori and FP-Growth, several other algorithms [6, 11, 17, 19] were designed to enhance the performance of ARM, which were applied in several real-world applications [4, 5, 10, 30].

A drawback of FIM and ARM is that they only consider the occurrence frequencies of itemsets. The other factors such as weights, interestingness, or unit profits of items are not considered to select more useful and meaningful patterns. High-utility itemset mining (HUIM) [32, 33] was developed to consider both the quantities and unit profits of item/sets to derive the set of high-utility itemsets (HUIs). To maintain the downward closure (DC) property and reduce the search space for mining HUIs, the transaction-weighted utilization (TWU) model [16] was designed. It provides the transaction-weighted downward closure (TWDC) property of the high transaction-weighted utilization itemsets (HTWUIs) to speed up the mining process and has become a standard mechanism for HUIM. Li et al. [18] then designed the isolated items discarding strategy (IIDS) to further reduce the number of candidates generated when mining HUIs using the TWU model. The incremental high-utility pattern (IHUP) algorithm [3] was developed to incrementally and interactively mine HUIs based on a tree structure similar to the FP-tree approach. The HUI-Miner algorithm [21] was proposed, which relies on a utility-list structure for mining HUIs without generating candidates and using a depth-first search. HUI-Miner constructs a vertical database representation to avoid performing multiple database scans while still deriving the set of high-utility *k*-itemsets, thanks to a simple join operation. To further improve the performance of HUIM, the FHM algorithm [7] was designed. It stores the relationships between all pairs of items (2-itemsets) to reduce the search space and prune a large amount of unpromising candidates early. Liu et al [26] designed the d2HUP algorithm with several pruning strategies to efficiently discover HUIs. Several algorithms [24, 28, 29] were also proposed for different applications and to address other issues related to HUIM. HUIM is a very active research area.

Although HUIM can reveal more useful information compared to FIM and ARM, the utilities of itemsets tend to increase if they contain more items. Thus, to provide a more fair measurement of the utilities of itemsets by considering their lengths, the task of high average-utility itemset mining (HAUIM) [13] was proposed. An itemset is considered to be a high average-utility itemset (HAUI) if its utility divided by its size (number of items that it contains) is no less than a user-specified minimum high average-utility threshold (count). The first two-phase TPAU algorithm [13] was designed and the average-utility upper bound (*auub*) model was developed to estimate the utilities of itemsets to ensure the completeness and the correctness of HAUIM. This model allows to reduce the search space by identifying a set of candidates called high average-utility upper-bound itemsets (HAUUBIs). Since the

TPAU algorithm employs a level-wise approach, it suffers from the drawback of having long execution times. To address this issue, the projection-based PAI algorithm [22] was developed to improve the mining performance using a novel pruning strategy. The high average-utility pattern (HAUP)-tree structure and its mining algorithm called HAUP-growth [20] were then designed to overcome the problem of multiple database scans of the TPAU algorithm. Each node in the HAUP-tree stores the quantities of its prefix items. Thus, HAUIs can be directly derived by performing combinations. This approach is efficient but it can consume a huge amount of memory if a database contains long transactions. Lu et al. presented the HAUI-tree approach [25] to mine HAUIs using an index table. A novel efficient HAUI-Miner algorithm [27] was then developed to mine HAUIs using a compact average-utility (AU)-list structure. The main problem of the HAUI-Miner algorithm is that it does not have very good scalability. Thus, it cannot be used to mine HAUIs in large-scale datasets. Besides, most of the previous HAUIM algorithms utilize the *auub* model to reduce the search space. Thus, unpromising itemsets with low relative average utility cannot be pruned early.

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 (a positive integer), denoted as $q(i_j, T_q)$. A profit table ptable = $\{p(i_1), p(i_2), \ldots, p(i_m)\}$ indicates the unit profit 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 1. It will be used in this paper as running example to illustrate the proposed approach step-by-step. It contains seven items denoted using the letters (a) to (g). The corresponding profit table is shown in Table 2, which indicates the unit profit of each item in that database.

TABLE I: A quantitative transactional database.

TID	Items with their quantities
T_1	*a*:1, *b*:2, *c*:4, *d*:3, *e*:8, *f*:2
T_2	*a*:3, *b*:3, *c*:8
T_3	*a*:2, *b*:5, *d*:5, *e*:7
T_4	*a*:4, *c*:4, *f*:2, *g*:12
T_5	*a*:5, *b*:2, *c*:3, *d*:5, *f*:8
T_6	*e*:1, *f*:1, *a*:3

TABLE II: A profit table.

Item	*a*	*b*	*c*	*d*	*e*	*f*	*g*
Profit	2	6	5	2	7	4	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 $q(i_j, T_q)$ is the purchase 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× 2) (= 2).

Definition 2 (Itemset utility in a transaction)**.** The utility of an itemset X in a transaction T_q is denoted as $u(X, T_q)$, and defined as:

$$u(X, T_q) = \sum_{i_j \in X \wedge X \subseteq T_q} u(i_j, T_q). \tag{2}$$

For example in Table I, the utility of the itemset $(abcd)$ in T_1 is calculated as $u(abcd, T_1)$ (= 1 × 2 + 2 × 6 + 4× 5 + 3× 2)(= 40).

Definition 3 (Itemset utility in *D*)**.** The utility of an itemset X in a database D is denoted as $u(X)$, and defined as:

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

For example in Table I, the utility of the itemset (ab) is calculated as $u(ab) = u(ab, T_1) + u(ab, T_2) + u(ab, T_3) + u(ab, T_5)$ (= 14 + 24 + 34 + 22)(= 94).

Definition 4 (Transaction utility)**.** The transaction utility of a transaction T_q is denoted as $tu(T_q)$, and defined as:

$$tu(T_q) = \sum_{i_j \in X} u(i_j, T_q). \tag{4}$$

For example in Table I, the utility of T_1 is calculated as $tu(T_1)$ (= 2 + 12 + 20 + 6 + 56 + 8)(= 104).

Definition 5 (Total utility in *D*)**.** The total utility of all transactions in a database D is denoted as TU, and defined as:

$$TU = \sum_{T_q \in D} tu(T_q). \tag{5}$$

For example in Table I, the total utility of the database is calculated as $TU = tu(T_1) + tu(T_2) + tu(T_3) + tu(T_4) + tu(T_5) + tu(T_6)$ (= 104 + 64 + 93 + 58 + 79 + 17) (= 415).

The above definitions are used in traditional HUIM. An itemset X is said to be a high-utility itemset (HUI) iff its utility in a database D is no less than a user-specified minimum high-utility count (minimum high-utility threshold multiplied by the total utility of the database), that is:

$$HUI \leftarrow \{X | u(X) \geq TU \times \delta\}, \tag{6}$$

where δ is the minimum high utility threshold.

In this example, assume that the minimum high-utility threshold is set to 14.7%. Hence, the minimum high-utility count is calculated as (415 × 14.7%)(= 61). The HUIs in the running example are thus: {(*ab*:94), (*abc*:135), (*abcd*:87), (*abcde*:96), (*abcdef*:104), (*ac*:121), (*ace*:78), (*acf*:121), (*bc*:117), (*bcd*:75), (*bcde*:94), (*bcdef*:102), (*c*:95) and (*ce*:76) (*e*:112)}.

In the above example, it is obvious that the utilities of itemsets tend to be larger for itemsets containing more items. The problem of high average-utility itemset mining (HAUIM) [27]

was thus proposed using a novel measure called average-utility, to provide a more fair measurement of the utilities of itemsets that takes their lengths into account. HAUIM is defined by the following definitions.

Definition 6 (Average-utility of an item in a transaction)**.** The average-utility of an item (i_j) in a transaction T_q is denoted as $au(i_j)$, and defined as:

$$au(i_j, T_q) = \frac{q(i_j, T_q) \times p(i_j)}{1} = \frac{u(i_j, T_q)}{1}. \quad (7)$$

For example in Table I, the average-utility of the item (a) in transaction T_1 is calculated as $au(a, T_1) = \frac{2}{1}$ = (= 2), which is equal to its utility in traditional HUIM.

Definition 7 (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:

$$au(X, T_q) = \frac{\sum_{i_j \in X \wedge X \subseteq T_q} u(i_j, T_q)}{|X| = k}. \quad (8)$$

For example in Table I, the average-utility of the itemset (ab) in transaction T_1 is calculated as $au(ab, T_1) = \frac{2+12}{2}$ (= 7).

Definition 8 (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:

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

For example in Table I, the average-utility of the itemset (ac) is calculated as: $au(ac)$ (= 11 + 23 + 14 + 12.5)(= 60.5).

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

$$HAUI \leftarrow \{X | au(X) \geq TU \times \delta\}. \quad (10)$$

To obtain a downward closure (DC) property for pruning the search space in HAUIM, the average-utility upper bound ($auub$) model [13] was designed to maintain the transaction-maximum utility downward closure (TMUDC) property of the high average-utility upper-bound itemsets (HAUUBIs). Definitions are given below.

Definition 9 (Transaction-maximum utility, *tmu*)**.** The transaction-maximum utility of a transaction T_q is denoted as $tmu(T_q)$, and defined as:

$$tmu(T_q) = max\{u(i_j, T_q) | i_j \subseteq T_q\}. \quad (11)$$

For example in Table I, $tmu(T_1)$ is calculated as $tmu(T_1)$ = $max\{2, 12, 20, 6, 56, 8\}$ (= 56), which is an upper-bound on the utility of any item in transaction T_1.

Based on the transaction-maximum utility, an average-utility upper-bound (*auub*) for itemsets is obtained to overestimate the average-utilities of potential candidate HAUIs. It is defined as follows.

Definition 10 (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:

$$auub(X) = \sum_{X \subseteq T_q \wedge T_q \in D} tmu(X, T_q). \quad (12)$$

For example in Table I, the average-utility upper-bound of the itemset (abc) in the database is calculated as: $auub(abc)$ (= 50 + 40 + 32) (= 128), which is no less than the minimum high average-utility count ($>$ 61). Thus, the itemset (*abc*) is considered as a candidate (a potential HAUI). Based on the *auub* model, the transaction-maximum utility downward closure (TMUDC) property was proposed. It defines a set of candidate HAUIs called the high average-utility upper-bound itemsets (HAUUBIs), which is defined as follows.

Definition 11 (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 value is no less than the minimum high average-utility count. The set of HAUUBIs is defined as:

$$HAUUBI \leftarrow \{X | auub(X) \geq TU \times \delta\}. \quad (13)$$

For example in Table I, the itemset (ab) is considered as a HAUUBI since its average-utility upper-bound is calculated as $auub(ab)$ = (177 $>$ 415$\times$ 14.7% = 61).

Theorem 1 (HAUIs $\subseteq$ HAUUBIs)**.** *The TMUDC property ensures that HAUIs $\subseteq$ HAUUBIs. Thus if an itemset is not a HAUUBI, then it is not a HAUI and none of its supersets is a HAUI.*

Although the *auub* model of the HAUUBIs provides the TMUDC property [13], which ensures that a correct and complete algorithm can be designed to mine the set of HAUIs, the upper-bound used to obtain HAUUBIs is loose. Thus, the search space can be huge for databases containing long transactions. In this paper, we thus present two tighter upper-bounds with three pruning strategies to reduce the overestimation of the average-utilities of itemsets. Thus the search space for discovering HAUIs can be greatly reduced and many unpromising candidates can be pruned early.

IV. Proposed Upper-Bounds and Pruning Strategies

In the past, many algorithms have been designed to mine high average-utility itemsets (HAUIs) [13, 20, 22, 25, 27]. The upper-bound *auub* model [13] was proposed to prune unpromising itemsets, which was used in many studies to improve the performance of HAUIM. However, the overestimations obtained in these studies using that model remain high. Thus, the search space for mining HAUIs based on the *auub* model remains huge. Besides, previous studies do not use efficient pruning strategies, which may lead to the problem of generating an exponential number of itemsets for dense databases or databases having long transactions. In this paper, two tighter upper-bounds and three pruning strategies are presented to efficiently reduce the search space for mining HAUIs.

A. The two proposed upper-bounds

In the *auub* model, the estimated upper-bound utility of an itemset *X* is defined as the sum of the transaction-maximum utilities (*tmu*) of transactions containing *X* in the database. Thus, the *auub* value is greatly influenced by items having very large utilities in each transaction. For example in Table I, the utility of items in transaction T_1 are $u(a)$(= 2), $u(b)$(= 12), $u(c)$(= 20), $u(d)$(= 6), $u(e)$(= 56) and $u(f)$(= 8). Thus, the *tmu* of T_1 is 56. However, this value is a very loose upper-bound on the average-utility of item (a). Thus, the item (a) may be considered as a HAUUBI during the mining progress. Because the *auub* model provides a loose upper-bound, the search space may be huge because supersets of (a) will be also considered. Thus, we present two novel upper-bounds as alternatives to the *auub* model. The goal is to reduce the number of unpromising itemsets considered for HAUIM. A modified average-utility (MAU)-list structure is developed to avoid performing multiple database scans and keep the required information in memory. The search space of the designed algorithm for mining HAUIs can be represented as an enumeration tree. Details are given below.

Definition 12 (Processing order)**.** The items in the enumeration tree and in transactions are sorted according to the *auub*-ascending order $\prec$, such that $auub(i_1) \prec auub(i_2) \prec \ldots \prec auub(i_n)$.

In the designed algorithm, the 1-itemsets having *auub* values less than the minimum high average-utility count are kept and sorted according to the above total order that will be used as processing order to explore the search space of itemsets. Based on previous studies [21], it was shown that choosing an appropriate processing order can reduce the size of the enumeration tree. In the given example of Tables I and II, the processing order is $\{e \prec f \prec d \prec c \prec b \prec a\}$.

After that, items having *auub* values that are smaller than the minimum high average-utility count are removed from transactions. This process may update the *tmu* values of transactions, and decrease the upper-bounds on the average-utilities of itemsets.

Definition 13 (Remaining maximal utility in a transaction, *remu*)**.** The remaining maximal utility of an itemset X in a sorted transaction is denoted as $remu(X, T_q)$, and defined as:

$$remu(X, T_q) = max\{u(i_{m+1}, T_q), u(i_{m+2}, T_q), \ldots, u(i_N, T_q)\}, \tag{14}$$

in which the number of items in the sorted transaction T_q is denoted as N, $X = \{i_1 \cup i_2 \cup i_3, \ldots, i_m\}$.

In the illustrated example, the remaining maximal utility of the itemset (dc) in the sorted transaction T_1 is calculated as $max\{u(b, T_1), u(a, T_1)\}$ = $max\{12, 2\}$(= 12).

Definition 14 (Looser upper-bound utility of an itemset in a transaction, *lub*)**.** The looser upper-bound utility of an itemset X in a sorted transaction T_q is denoted as $lub(X, T_q)$, and defined as:

$$lub(X, T_q) = \frac{u(X, T_q) + |X| \times remu(X, T_q)}{|X|}. \tag{15}$$

The *lub* is an upper-bound on the average-utility of an itemset X in a transaction T_q, which can be less than the *tmu* value. For example, the *lub* of the itemset (dc) in the sorted transaction T_1 is calculated as $lub(dc, T_1)$ (= 13 + 12) (= 25), which is less than the transaction-maximum utility in T_1 ($tmu(T_1)$(= 56)).

Definition 15 (Looser upper-bound utility of an itemset in a database)**.** The looser upper-bound utility of an itemset X in a sorted database is denoted as $lub(X)$, and defined as:

$$lub(X) = \sum_{X \in T_q \wedge T_q \subseteq D} lub(X, T_q). \tag{16}$$

For example, the looser upper-bound of the itemset (dc) in the illustrated database is calculated as $lub(dc)$ (= 13 + 12) + (12.5 + 12)(= 49.5), which is much less than the upper-bound value obtained using the traditional *auub* model, i.e. $auub(dc)$(= 56 + 32)(= 90).

Compared with the *auub* model, the designed *lub* model can prune many unpromising itemsets early since the *tmu* value of each transaction may be greatly reduced. Based on the *lub* model, the following property holds to ensure the completeness and correctness of the designed algorithm based on the *lub* model.

Property 1 (Anti-monotonicity property of *lub*)**.** Let $Y(= i_j \cup \cdots \cup i_m \cup i_{m+1} \cup \cdots \cup i_n)$ be a superset of X $(= i_j \cup \cdots \cup i_m)$ such that $X \subseteq Y$. Thus, we can obtain that $au(Y) \leq lub(X)$.

Proof. Let T_q be a sorted transaction containing both *X* and *Y*, and let $tids(X)$ and $tids(Y)$ be the sets of transactions containing *X* and *Y*, respectively. Furthermore, let $Y \backslash X$ be the remaining items of Y without *X*. We can obtain that:

$au(Y, T_q) - lub(X, T_q)$ = $\frac{u(X,T_q)}{|Y|}$ - $\frac{u(X,T_q)+|X|\times remu(X,T_q)}{|X|}$

=$\frac{u(X,T_q)+u(Y\backslash X,T_q)}{|Y|}$ - $\frac{u(X,T_q)+|X|\times remu(X,T_q)}{|X|}$

= $\frac{u(X,T_q)}{|Y|}$ - $\frac{u(X,T_q)}{|X|}$ + $\frac{u(Y\backslash X,T_q)}{|Y|}$ - $remu(X, T_q)$

$\leq$ $\frac{u(X,T_q)}{|X|}$ - $\frac{u(X,T_q)}{|X|}$ + $\frac{u(Y\backslash X,T_q)}{|Y|}$ - $remu(X, T_q)$

$\leq$ $\frac{u(Y\backslash X,T_q)}{|Y|}$ - $remu(X, T_q)$

=$\frac{u(i_{m+1},T_q)+\ldots+u(i_n,T_q))}{|Y|}$ - $max(u(i_{m+1}, T_q), \ldots, u(i_N, T_q)$

$\leq$ 0,

where *N* is the length of the transaction T_q, and $n \in \{1, \ldots, N\}$

Since $tids(Y) \subseteq tids(X)$, thus $\sum_{T_q \in tids(Y)} au(Y, T_q) \leq \sum_{T_q \in tids(Y)} lub(X, T_q) \leq \sum_{T_q \in tids(X)} lub(X, T_q) = lub(X)$. □

Using a depth-first search, a (*k*+1)-itemset (child node) can be obtained by combining two *k*-itemsets. For example, the itemset (*fd*) can be formed by combining the itemsets (*f*) and (*d*), and the itemset (*fdc*) can be formed by combining the itemsets (*fd*) and (*fc*). Using this approach and based on the above property, unpromising (*k*+1)-itemsets can be pruned early if the *lub* value of a *k*-itemset is less than the minimum high average-utility count. Thus, we can obtain the following theorem.

Theorem 2 (Looser upper-bound model, *lub*)**.** *If the* $lub(X)$ *value of an itemset* X *is less than the minimum high average-utility count, any extension* Y *of* X *is not a HAUI. In other*

words, all extensions of the itemset X can be ignored while ensuring that all HAUIs can still be found, thus preserving the completeness and correctness of the designed algorithm.

The proposed *lub* model is very useful to prune unpromising itemsets. However, the search space can still be very large due to the utilities of irrelevant items that are still summed up to calculate the *tmu* values of transactions. Thus, it is necessary to develop another tighter upper-bound model to prune the search space efficiently. The details are as follows.

Definition 16 (Irrelevant itemsets in the database)**.** An irrelevant itemset is an item having an *auub* value that does not satisfy the minimum high average-utility count. Hence, it cannot be used to extend an itemset to generate promising itemsets.

In the running example, the processing oder is $\{e \prec f \prec d \prec c \prec b \prec a\}$. Thus item (*e*) can be extended with any of the items succeeding (*e*) such as (*f*), (*d*), (*c*), (*b*), and (*a*). Simillarly, the item (*f*) can be extended with any of the items succeeding (*f*), such as (*d*), (*c*), (*b*), and (*a*). The reason is that all these items having *auub* values that are less than *auub*(i_j) are irrelevant, and can be ignored without missing any HAUIs. Thus, the transaction-maximum utility (*tmu*) can be revised and reduced by temporally removing irrelevant items.

Definition 17 (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:

$$rmu(X, T'_q) = max\{u(i_1, T_q), u(i_2, T_q), \ldots, u(i_{|T_q|}, T_q)\}, \tag{17}$$

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 for the revised transactions. Besides, since irrelevant items after X are temporary removed to obtain the updated transaction-maximum utility of the itemset X, the $rmu(X, T'_q) \leq tmu(X, T_q)$ preserves the correctness and completeness for mining HAUIs. Based on this definition, we can further lower the upper-bounds on itemsets as follows.

Definition 18 (Revised tighter upper-bound model, *rtub*)**.** The revised tighter upper bound of an itemset *X* is denoted as $rtub(X)$, and defined as:

$$rtub(X) = \sum_{X \subseteq T'_q \in D} rmu(X, T'_q). \tag{18}$$

In the running example, suppose that the itemset (*e*) and its supersets have been processed. The itemset (*e*) has become an irrelevant item, which can be temporally removed from the original database. By performing this removal, $rmu(T_1)(= 20) < tmu(T_1)(= 56), rmu(T_3)(= 30) < tmu(T_3)(= 49), rmu(T_6)(= 6) < tmu(T_6)(= 7)$. Therefore, the *rtub* values of itemsets contained in the revised transactions can be greatly reduced. For example, $rtub(fd)$ is calculated as $rtub(fd)$(= 20 + 32)(= 52), which is less than $auub(fd)$(= 56 + 32)(= 88) after the irrelevant items are temporally removed. To facilitate the efficient removal of irrelevant items, a modified average-utility (MAU)-list structure is designed to avoid performing multiple database scans. Based on the *rtub*, the following theorem can be obtained.

Theorem 3. *For an itemset X, it holds that $au(X) \leq rtub(X) \leq auub(X)$.*

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

Thus, the proposed *rtub* model is a tighter upper-bound compared to the *auub* model. Hence, the search space can be greatly reduced. Since supersets of itemsets are generated by merging two itemsets with the same (*k*-1)-itemset as prefix, using this theorem can greatly reduce the search space if these unpromising supersets can be identified early. The anti-monotonicity property of *rtub* is defined as follows.

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

Proof. Since $X \subseteq Y$, it follows that $tids(Y) \subseteq tids(X)$. 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 than the minimum high average-utility count, its superset 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 the performance of mining HAUIs can be greatly improved.

To avoid performing multiple database scans and efficiently removing temporal irrelevant items, a modified average utility (MAU)-list structure is designed to keep the remaining maximal utility and revised transaction-maximum utility of each transaction. Details are given below.

Definition 19 (Modified average-utility (MAU)-list structure)**.** The MAU-list of an itemset *X* keeps the relevant information about this itemset in transactions where it appears. Each entry of a MAU-list has values for four fields: 1). the transaction ID of a transaction where *X* appears; 2). the utility of *X* in T_q (*iutil*); 3). the revised maximal utility of (*X*) in transaction T_q (*rmu*) and 4). the remaining maximal utility of *X* in T_q (*remu*).

In this example, the MAU-list of each 1-itemset is constructed based on the *auub*-ascending order. The results are shown in Figure 1. Note that the proposed MAU-list structure is different from the AU-list structure used in HAUI-Miner [27] since the designed MAU-list stores different information. The MAU-list structure can be efficiently obtained by performing a single database scan.

B. Proposed algorithm with pruning strategies

In the past, list structures have been used in algorithms such as FHM [7] and HAUI-Miner [21] to efficiently discover HUIs and HAUIs, respectively. The list structure of a *k*-itemset ($k > 1$) is constructed by performing join operations using (*k*-1)-list structures of some of its subsets. However, as analyzed

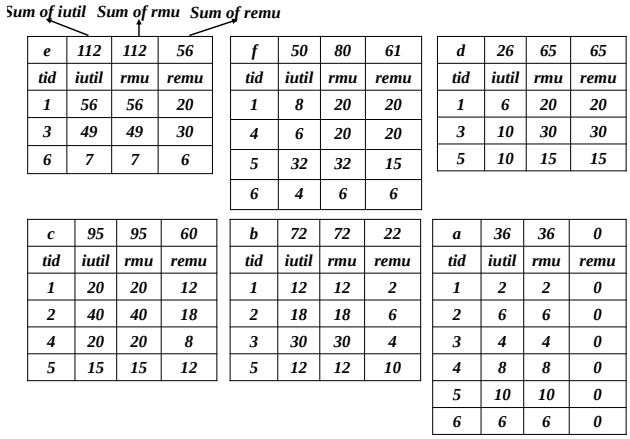

Fig. 1: The MAU-list structure.

in FHM and d2HUP [26], performing the join operation is costly. Thus, the designed algorithm may have inefficiency and scalability problems. To deal with this issue and further reduce the search space, a more efficient high average-utility pattern mining (EHAUPM) algorithm is designed and three pruning strategies are presented to reduce the search space for mining HAUIs.

Based on the *auub* upper bound, it can be obtained that if the upper-bound on the average-utility of an itemset is less than the minimum high average-utility count, all its extensions do not need to be considered since they cannot be HAUIs. Thus, the first pruning strategy is defined as follows.

Pruning Strategy 1 (Remove unpromising items from database, RUI): The unpromising itemsets (1-items not satisfying the minimum high average-utility count) are removed from the database.

Proof. According to the anti-monotonicity of *auub* [13], this property holds and preserves the correctness and completeness for mining HAUIs. □

This pruning strategy can be used to lower the upper-bounds for 1-items by performing a single database scan. After that, the transaction-maximum utilities of items in each transaction may decrease if an unpromising item has the maximal utility in that transaction. Thus, the remaining items in the database are all promising. Hence, any itemset appearing in that modified database does not contain unpromising items and many join operations can be avoided by not constructing the list structures of itemsets containing unpromising items. To efficiently check unpromising 2-itemsets (subsets) for a *k*-itemset ($k > 2$), the compact estimated co-occurrence of average-utility matrix (EAUCM) structure is designed as follows.

Definition 20 (Estimated average-utility co-occurrence matrix, EAUCM)**.** The EAUCM is a matrix that stores the *auub* values of 2-itemsets, in which each 1-itemset in the matrix is a 1-HAUUBI.

Thus, it is possible to identify some unpromising *k*-itemsets ($k \geq 3$) by looking at the 2-itemsets that they contains. Based on the anti-monotonicity of the *auub* model, it can be ensured that if the *auub* value of a 2-itemset is no greater than that of the minimum high average-utility count, all its supersets cannot be HAUIs.

Pruning Strategy 2 (Estimated co-occurrence of average-utility pruning strategy, ECAUPS): If the *auub* value of a 2-itemset in the EAUCM is less than the minimum high average-utility count, any of its supersets can be directly ignored.

Proof. According to the *auub* model, $au(X) \leq auub(X)$, and $auub(Y) \leq auub(X)$ if $X \subseteq Y$. Thus, all unpromising 2-itemsets are not HAUUBIs, as well as their supersets. They can thus be directly pruned from the search space. □

For the running example, the EAUCM is presented in Table III.

TABLE III: The EAUCM of the running example.

	e	*f*	*d*	*c*	*b*
f	63	-	-	-	-
d	105	88	-	-	-
c	56	108	88	-	-
b	105	88	137	128	-
a	112	115	137	148	177

The first pruning strategy is only applied during the initial database scan. The second pruning strategy is repeatedly applied before generating any new *k*-itemset (k>1) and the designed MAU-list structure can be used to avoid performing unnecessary computation. However, the search space can still be reduced. A similar idea was designed in the HUP-Miner algorithm [15] to discover HUIs. In the pruning strategy 3, the designed *lub* and *rtub* models are used to reduce the search space while generating the (k+1)-itemsets by performing join operations for MAU-lists of *k*-itemsets.

Pruning Strategy 3 (Stop unpromising join operations, SUJ): Let P_x and P_y be two itemsets. The join operation of these itemsets does not need to be completed if any of the following equations is equal to a value that is less than the minimum high average-utility count, while performing the join operation:

$$SUJA(P_x) = auub(P_x) - \sum_{T_q \in tids(P_x) \wedge T_q \notin tids(P_y), T_q \in D} tmu(T_q). \quad (19)$$

$$SUJL(P_x) = lub(P_x) - \sum_{T_q \in tids(P_x) \wedge T_q \notin tids(P_y), T_q \in D} au(P_x, T_q) + remu(P_x, T_q). \quad (20)$$

$$SUJR(P_x) = rtub(P_x) - \sum_{T_q \in tids(P_x) \wedge T_q \notin tids(P_y), T_q \in D} rmu(P_x, T_q). \quad (21)$$

Proof. Since the *tidset* of an itemset P_{xy} is the union of the *tidsets* of two itemsets P_x and P_y, the transactions containing P_x but not P_y do not contain the itemset P_{xy} and its supersets. During the join operation, transactions containing P_x can thus be removed, and this will not affect the HAUIs found by the algorithm. According to the anti-monotonicity of *auub*, the join operation can be skipped when at least one of the values among $SUJL(P_x), SUJR(P_x)$, and $SUJA(P_x)$ is less than the minimum high average-utility count. Since the calculations of $SUJL(P_x), SUJR(P_x)$ and $SUJA(P_x)$ are performed during join operations, several unnecessary join operations can

thus be avoided, which can greatly reduce the cost of mining HAUIs. □

This pruning strategy is useful to reduce the search space since it can avoid performing many unnecessary costly join operations. The two proposed upper-bounds and three pruning strategies are integrated in the proposed efficient high average-utility pattern mining (EHAUPM) algorithm, presented in Algorithm 1.

Algorithm 1: EHAUPM Algorithm

Input: D, a transactional database; $ptable$, a profit table; δ, a minimum high average-utility threshold.
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 the *auub* of each item in *D*;
2 calculate the total utility *TU* in *D*;
3 **for** *each item* i_j *such that* $auub(i_j) < TU \times \delta$ **do**
4 remove i_j from D;
5 recalculate the *auub* of each remaining item in *D*;
6 sort items in transactions in *auub*-ascending order;
7 construct $I^*.AULs$ and *EAUCM*;
8 **Search**($\emptyset$, $I^*.AULs$, *EAUCM*, δ, *TU*);
9 return HAUIs;

The proposed algorithm first calculates the *auub* of each item (Line 1) and the total utility (*TU*) by scanning the database once (Line 2). For each item in the database, its *auub* value is then checked against the minimum high average-utility count (Line 3), and items not satisfying the threshold are removed (Line 4). After that, the *auub* value is then calculated again for each item to obtain its lower upper-bound value (Line 5). The remaining items in the transactions are sorted by their *auub*-ascending order (line 6), and the MAU-list of each remaining item is then built (Line 7), as well as the EAUCM of 2-itemsets (Line 7). After that, the search algorithm is then applied to recursively mine HAUIs using $I^*.AULs$ by performing a depth-first **Search** (Line 8), and the resulting HAUIs are returned (Line 9). The pseudocode of the **Search** algorithm is given in Algorithm 2.

As shown in Algorithm 2, the **Search** algorithm takes an itemset and a set of its 1-extensions as arguments and then sequentially processes each itemset (X_a) in the set of 1-extensions of itemset (*X*) (Lines 1 to 12). For each itemset (X_a), the utility of each entry in the MAU-list structure of X_a is then summed (Line 2), and if its value is no less than the minimum high average-utility count, X_a is a high average-utility itemset (Line 3). After that, the minimum value between the looser upper-bound value (*lub*) and the revised tighter upper-bound value (*rtub*) is compared with the minimum high average-utility count to determine whether the supersets of X_a should be processed by the depth-first search (Line 5). If the condition is passed, the set of MAU-lists of all 1-extensions

Algorithm 2: Search Algorithm

Input: *X*, an itemset; *extensionsOfX*, a set of MAU-lists of all 1-extensions of *X*; *EAUCM*, an estimated average-utility co-occurrence matrix; δ, a minimum high average-utility threshold; TU, a total utility.
Output: The set of high average-utility itemsets, HAUIs.

1 **for** *each* $X_a \in extensionsOfX$ **do**
2 **if** $\frac{sum(X_a.iutils)}{|X_a|} \geq TU \times \delta$ **then**
3 $HAUIs \leftarrow HAUIs \cup X_a$;
4 **if** $min\{lub(X_a), rtub(X_a)\} \geq TU \times \delta$ **then**
5 $extensionsOfX_a \leftarrow \phi$;
6 **for** *each* $X_b \in extensionsOfX$ *such that* $a \prec b$ **do**
7 **if** $EAUCM(X_a, X_b) \geq TU \times \delta$ **then**
8 $X_{ab} = X_a \cup X_b$;
9 $Xab \leftarrow$ **Constrcut** (X, X_a, X_b);
10 **if** $X_{ab}.AULs \notin null$ **then**
11 $extensionsOfX_a \leftarrow extensionsOfX_a \cup X_{.ab}.AUL$;
12 **Search**(X_a, $extensionsOfX_a$, $EAUCM$, δ, TU);
13 return HAUIs;

of X_a is constructed by extending each itemset (X_b) such that $a \prec b$ (Lines 6 to 11). To avoid performing costly join operations for unpromising itemsets, each 1-extension of X_a will be checked against the minimum high average-utility count using the EAUCM before constructing its MAU-list (Lines 7 to 11). If the MAU-list of X_{ab} is built, $extensionsOfX_a$ is then updated for further processing (Lines 8 to 11) by recursively executing the Search algorithm (Line 12). After that, the set of HAUIs is returned. The **Construct** algorithm is shown in Algorithm 3.

First, the MAU-list of X_{ab} is initialized as an empty list (Line 1), and the variables $lubOfX_a$ and $rtubOfX_a$ are respectively set to $lub(X_a)$ and $rtub(X_a)$ (Line 2). For each entry E_a in $X_a.AUL$, the algorithm identifies the entry E_b having the same transaction ID as E_a (Line 4). If there are no such entries, the pruning strategy 3 is applied to avoid constructing the MAU-list of an unpromising itemset (Lines 11 to 14). Otherwise, a new entry E_{ab} is built (Line 7). If the MAU-list of X is empty, it indicates that the transaction contains X_a and X_b but not X, and the new entry E_{ab} is built (Line 9). Note that the *rmu* and *remu* of E_{ab} should be respectively assigned to the same value of E_a and E_b. Since $a \prec b$ and all the items in the transactions are sorted according to the *auub*-ascending order ($\prec$), the remaining maximal utility of X_{ab} is the same as that of X_b in the corresponding transactions containing both X_{ab} and X_b.

V. An Illustrated Example

In this section, a simple example is presented using the database depicted in Tables I and II to show how the proposed

Algorithm 3: Construct Algorithm

Input: X, an itemset; X_a, the extension of X with an item a;X_b, the extension of X with an item b; δ, the minimum high average-utility threshold; TU, the total utility.

Output: X_{ab}, the MAU-list of X_{ab}.

```
1  set X_ab.AUL ← ϕ;
2  set lubOfX_a = lub(X_a), rtubOfX_a = rtub(X_a);
3  for each entry E_a ∈ Xa.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, E_b.remu >;
8          else
9              E_ab ←< E_a.tid, E_a.iutil +
                 E_b.iutil, E_a.rmu, E_b.remu >;
10     else
11         lubOfX_a = lubOfX_a − E_a.iutil/|E_a| − E_a.remu;
12         rtubOfX_a = rtubOfX_a − E_a.remu;
13         if min{lubOfX_a, rtubOfX_a} < TU × δ then
14             return ∅;
15 return X_ab.AUL;
```

algorithm is applied step-by-step. The minimum high average-utility threshold is initially set to 14.7% (it is specified by the user based on his preferences). First, the *auub* of each item and the total utility (*TU*) in *D* are respectively calculated by scanning the database once. The *auub* values of 1-items are shown in Table IV and the *TU* is calculated as 415.

TABLE IV: The *auub* values of items.

Item	*a*	*b*	*c*	*d*	*e*	*f*	*g*
auub	208	177	152	137	112	121	24

After that, the *auub* of each item is compared with the minimum high average-utility count. The database is then scanned again to remove unpromising itemsets, and the transaction-maximum utility (*tmu*) of each transaction is calculated. This process can lower the upper-bound value of the itemset, which can be used to reduce the search space for mining the HAUIs. For example, the *auub* of the item (*a*) becomes *auub*(*a*)(= 204); the *auub* of the item (*c*) becomes *auub*(*c*) (= 148), and the *auub* of the item (*f*) becomes *auub*(*f*)(= 117). In the running example, the item (*g*) is removed since its *auub* value is less than the minimum high average-utility count (= 415× 0.147 = 61.005).

After that, all the remaining items are sorted in *auub*-ascending order ($e \prec f \prec d \prec c \prec b \prec a$). The depth-first search is then performed for mining the set of HAUIs. In this example, the item (*e*) is first processed and its looser upper-bound (*lub*) and remaining tighter upper-bound (*tub*) values are respectively calculated as *lub*(*e*) (= 56 + 20 + 49 + 30 + 7 +6)(= 168) and *rtub*(*e*)(= 56 + 49 + 7)(= 112), which are no less than the minimum high average-utility count (> 61.005). Thus, the succeeding items (*f*), (*d*), (*b*), and (*a*) are successively appended to the item (*e*) to construct their MAU-lists using the **Construct** algorithm. However, the item (*c*) will not be used to construct an MAU-list for the itemset (*ec*) since the co-occurrence average-utility of (*ec*) is less than the minimum high average-utility count. In the running example of (*ef*), the entries of the MAU-list of (*e*) and (*f*) are compared to find those having the same *tids*. Those entries are used for generating entries for the MAU-list of (*ef*). If such entry does not exist, the third strategies (*SUJL* and *SUJR*) are then applied to stop the construction process for unpromising itemsets. In the example, the itemset (*ef*) is not pruned by the *SUJL* and *SUJR* strategies since *rtub*(*ef*)(= 63) and *lub*(*ef*)(= 63.5), which are larger than the minimum high average-utility count. The depth-first search is then recursively continued to consider the supersets of (*ef*) until all itemsets in the search space have been considered. The enumeration trees of items (*e*), (*f*), and (*d*) are shown in Figure 2 and 3.

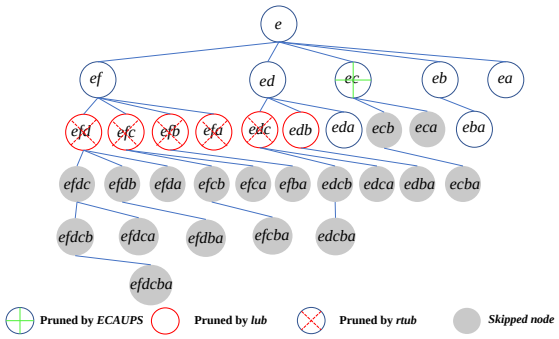

Fig. 2: The enumeration tree of the item (*e*).

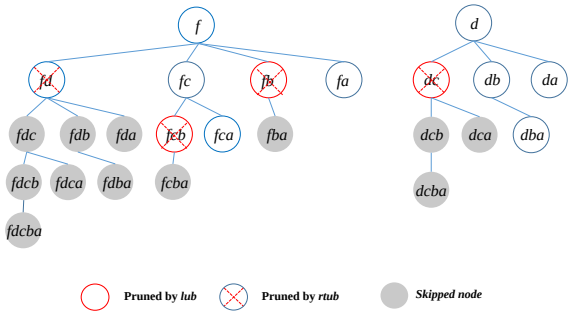

Fig. 3: The enumeration trees of the items (*f*) and (*d*).

In this example, it is obvious that the two proposed upper-bounds are efficient for pruning unpromising itemsets and reducing the search space. For instance, without *rtub* and *lub*, the itemset (*fd*), (*edb*), and their supersets could not be pruned early since $auub(fd)$(= 90), and $auub(edb)$(= 105). However, based on the designed models, $rtub(fd)$(= 52) and $rtub(edb)$(= 60.3). Thus, these itemsets can be pruned early and their supersets can be skipped. Besides, the ECAUPS also plays an important role for pruning unpromising itemsets. For example, the itemset (*ec*) and its supersets are pruned early

(before constructing their MAU-lists). This process is then repeatedly performed until no candidates are generated. The final set of HAUIs is {*e*:112, *ec*:76, *eb*:73.5, *c*:95, *b*:72}.

VI. Experimental Evaluation

In this section, the performance of the proposed EHAUPM algorithm is compared with the state-of-art HAUI-Miner [27] algorithm. Moreover, the designed pruning strategies are individually evaluated in terms of effectiveness. The compared algorithms are listed in Table V. Five algorithms are compared to evaluate the efficiency and effectiveness of the designed pruning strategies and two upper-bounds.

TABLE V: The compared algorithms.

Algorithm	Upper-bound	Pruning strategies
HAUI-Miner	*auub*	-
HAUI-Miner1	*auub*	ECAUPS
HAUI-Miner2	*auub*	ECAUPS+RUI
HAUI-Miner3	*auub*	ECAUPS+RUI+SUJA
EHAUPM	*rtub+lub*	ECAUPS+RUI+SUJL+SUJR

The algorithms are implemented in Java and executed on a computer equipped with an Intel(R) Core(TM) i7-4790 3.60GHz processor and 8 GB of main memory, running the 64 bit Microsoft Windows 7 operating system. Experiments were conducted on six different datasets, including five real-world datasets [8] and a synthetic dataset generated using the IBM Quest Synthetic Data Generator [14]. A simulation model [16] was developed to generate the quantities and profit values of items in transactions for all datasets. A log-normal distribution was used to randomly assign quantities in the [1,5] interval and item profit values in the [1,1000] interval. Parameters of the datasets and their characteristics are summarized in Table VI and Table VII, respectively.

TABLE VI: Parameters of the datasets.

$\#\|\mathbf{D}\|$	Total number of transactions
$\#\|\mathbf{I}\|$	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	$\#\|\mathbf{D}\|$	$\#\|\mathbf{I}\|$	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	Dense
chess	3,196	76	37	37	Dense
mushroom	8,124	120	23	23	Dense
T40I10D100K	100,000	1,000	39.6	77	Dense

A. Runtime

In this subsection, the runtimes of the algorithms are compared for different minimum high average-utility threshold values. Results are shown in Figure 4.

It can be observed in Figure 4 that the proposed strategies provides a considerable performance improvement compared to the traditional state-of-the-art HAUI-Miner algorithm. Generally, the proposed algorithm is about up to five or six

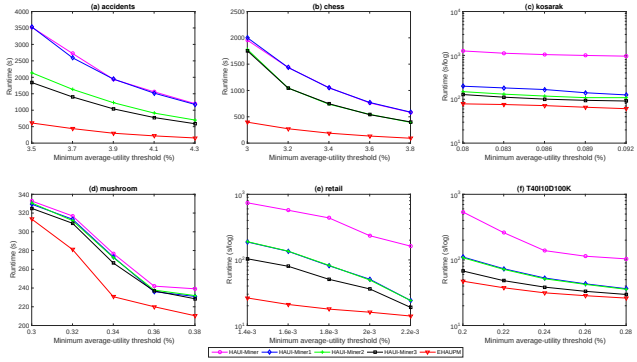

Fig. 4: Runtimes for various high minimum average-utility thresholds.

orders of magnitude faster than the HAUI-Miner algorithm. For example in Figure 4(c), when the minimum high average-utility threshold is set to 0.08%, the HAUI-Miner, the HAUI-Miner1, HAUI-Miner2, HAUI-Miner3, and EHAUPM algorithms terminate in 1,262 seconds, 198 seconds, 149 seconds, 127 seconds, and 78 seconds, respectively. Thus, the proposed strategies can be used to efficiently reduce the runtime for mining the set of HAUIs. Besides, it also indicates that the two proposed tighter upper-bounds can be used to efficiently and effectively prune unpromising candidates early for mining the HAUIs. Moreover, each strategy can be individually applied in the proposed algorithm. Thanks to the ECAUPS, a larger number of unpromising candidates can be pruned compared to the *auub* model, but the ECAUPS may be unable to prune unpromising candidates when the minimum high average-utility threshold is set too low. In Figure 4(a), it also can be observed that when the *SUJA* strategy is adopted, the runtime is about 1,400 seconds less than when using the traditional *auub* model. The reason is that as many unpromising candidates are identified, performing the costly join operation becomes unnecessary for their supersets. In summary, the proposed algorithm with its pruning strategies outperforms the traditional HAUI-Miner algorithm since its search space can be considerably reduced by adopting the tighter upper-bounds.

B. Memory usage

In this section, we compare the memory usage of the algorithms. The peak memory usage has been measured by utilizing the standard Java API for the six datasets. The results for various minimum high average-utility threshold values on the six datasets are shown in Figure 5.

In Figure 5, it can be observed that the memory usage of the compared algorithms steadily decreases as the minimum high average-utility threshold is increased. This is reasonable since when the minimum high average-utility threshold is set higher, fewer itemsets are HAUIs. Thus, the search space is reduced, as well as the memory usage. In addition, the memory usage can be further reduced when more pruning strategies are applied in the algorithm. The reason is that a large number of unpromising candidates can be pruned using the proposed upper bound models except for the retail dataset. The reason

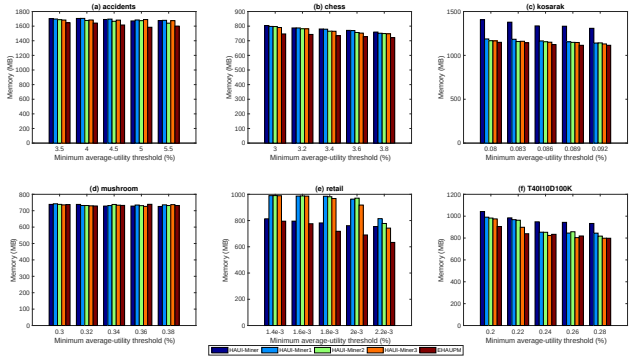

Fig. 5: Number of candidates for various minimum high average-utility threshold values.

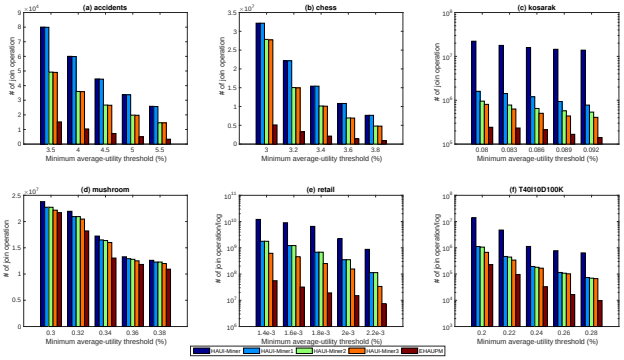

Fig. 6: Number of join operation for various minimum high average-utility thresholds.

is that the ECAUPS has to keep the information about 2-itemsets in memory. Thus, the proposed algorithm sometimes requires more memory than the traditional HAUI-Miner algorithm. However, when the proposed tighter upper-bounds are applied, the memory usage dramatically decreases since a huge number of unpromising candidates are pruned. It also can be observed that the memory usage slowly decreases when more pruning strategies are applied. The reason is that the proposed algorithm stores more information in its list structure such as the remaining maximal utility (*remu*). It thus indicates that more memory is required compared to the traditional HAUI-Miner algorithm. Thanks to the designed tighter upper-bounds and the pruning strategies, the memory usage decreases when the upper-bounds are applied with the pruning strategies. Particularly, for the mushroom dataset in Figure 5(d), all the compared algorithms require a constant amount of memory when the minimum high average-utility threshold is varied. For the chess, retail and T40I10D100K datasets, the memory usage decreases when the pruning strategies are applied in the designed algorithm. Hence, the memory usage of the proposed algorithm is considerably less than that of the state-of-the-art HAUI-Miner algorithm.

C. Number of join operations

In this section, the number of join operations performed by the algorithms is compared for various high minimum-average utility threshold values. The results are shown in Figure 6.

In Figure 6, it can be observed that the number of join operations performed by the proposed algorithms is considerably less than that of the HAUI-Miner algorithm. HAUI-Miner performs the largest number of join operations, which indicates that the search space of HAUI mining is very large. Based on the proposed strategies and the tighter upper-bounds, the number of join operations can be dramatically reduced. For example, for the kosarak and T40I10K100K datasets, EHAUPM has the best performance compared to the other algorithms since the number of join operations is considerably less. Also, it can be seen that the designed upper-bounds have better performance for the accident, chess, kosarak, T40I10D100K and retail datasets. The reason is that different datasets have different characteristics. Thus, different strategies may lead to different effects in terms of unpromising candidate pruning. When all the designed pruning strategies and two upper-bounds are applied, a large number of unpromising itemsets can be pruned from the search space. Thus, it can be concluded that the designed pruning strategies can greatly decrease the number of join operations for constructing the MAU-lists. Besides, the proposed upper-bounds are efficient to decrease the upper-bound values of itemsets. As a result, the search space for mining the HAUIs can be greatly reduced.

D. Scalability

The scalability of all algorithms was evaluated by measuring the runtime, memory usage and number of join operations performed by the algorithms on several datasets denoted as T10I4N4KD$|X|K$, where X indicates the dataset size, which was varied from 100(K) to 500(K) transactions, using an increments of 100(K) transactions. The minimum high average-utility threshold for all datasets in this experiment was set to 0.0015%. The results are shown in Figure 7.

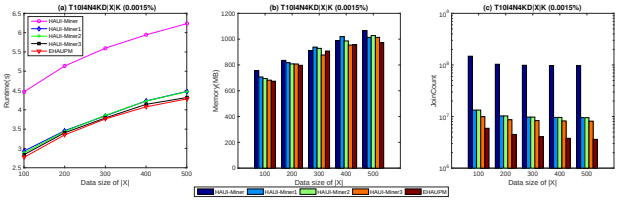

Fig. 7: Scalability when dataset size is varied.

In Figure 7, it can be observed that all the compared algorithms employing the proposed pruning strategies scale well as the dataset size is increased, and that the proposed EHAUPM algorithm has the best performance among all the compared algorithms. When dataset size increases, the runtime and memory usage both increase for all compared algorithms, while the number of join operations decreases. The reason is that the MAU-list of each promising itemset has to keep more information. When dataset size increases, more time is required to perform the join operations. Meanwhile, a larger dataset size means a higher total utility, as well as a higher minimum high average-utility count. Thus, less join operations are needed, as shown in Figure 7(c). From these

results, it can be seen that the number of join operations can be steadily reduced by applying the developed pruning strategies, which correspondingly lead to decrease the runtime and memory usage. Besides, the gap in terms of runtime between the compared algorithms increases with dataset size. Hence, the proposed upper-bounds and pruning strategies are more efficient than those used by the traditional HAUI-Mine algorithm using the *auub* model. From the experiments, it can be concluded that the proposed algorithm outperforms the state-of-the-art HAUI-Miner algorithm for discovering HAUIs in terms of runtime, memory usage and the number of join operations for various dataset sizes.

VII. Conclusion

In the past, the *auub* model was adopted in HAUIM to provide an upper-bound on the utilities of itemsets for maintaining the downward closure property. Because the auub model is based on the maximal utility in each transaction, it is greatly influenced by the utility values of items in transactions. In this paper, we present two efficient upper-bounds to further reduce the upper-bound values compared to the traditional *auub* model. The MAU-list structure is also developed to keep the necessary information for the mining process, and avoid performing multiple database scans. Three pruning strategies are respectively developed to prune unpromising itemsets early. Thus, the computational time and search space can be greatly reduced. Experiments on both real-life and synthetic datasets showed that the proposed algorithm significantly outperforms the state-of-the-art HAUI-Miner algorithm and the developed pruning strategies are efficient to speed up the mining performance, and can also be used with the traditional *auub* model.

Acknowledgment

This research was partially supported by the National Natural Science Foundation of China (NSFC) under Grant No.61503092, and by the Tencent Project under grant CCF-Tencent IAGR20160115.

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] 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 of Knowledge and Data Engineering*, vol. 21, no. 12, pp. 1708-1721, 2009.

[4] M. S. Chen, J. S. Park, and P. S. Yu, "Efficient data mining for path traversal patterns," *IEEE Transactions of Knowledge and Data Engineering*, vol. 10, no. 2, pp. 209-221, 1998.

[5] C. Creighton and S. Hanash, "Mining gene expression databases for association rules," *Bioinformatics*, vol. 19, no. 1, pp. 79-86, 2003.

[6] C. Chi and K. Lain, "A new fp-tree algorithm for mining frequent itemsets," *The Content Computing, Advanced Workshop on Content Computing*, pp. 266-277, 2004.

[7] P. Fournier-Viger, C. W. Wu, S. Zida, and V. S. Tseng, "FHM: faster high-utility itemset mining using estimated utility co-occurrence pruning" *Foundations of Intelligent Systems*, vol. 8502, pp. 83-92, 2014.

[8] 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.

[9] P. Fournier-Viger, J. C. W. Lin, R. U. Kiran, Y. S. Koh, and R. Thomas, "A survey of sequential pattern mining," *Data Science and Pattern Recognition*, vol. 1(1), pp. 54-77, 2017.

[10] E. Georgii, L. Richter, U. Rckert, and S. Kramer, "Analyzing microarray data using quantitative association rules," *Bioinformatics*, vol. 21 Suppl 2, no. Suppl 2, pp. ii123-ii129, 2005.

[11] G. Grahne and J. Zhu, "Fast algorithms for frequent itemset mining using fp-trees," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, pp. 1347-1362, 2005.

[12] 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.

[13] T. P. Hong, C. H. Lee, and S. L. Wang, "Effective utility mining with the measure of average utility," *Expert System with Applications*, 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] 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.

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

[18] 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.

[19] C. W. Lin, T. P. Hong, and W. H. Lu, "The Pre-FUFP algorithm for incremental mining," *Expert Systems with Applications*, vol. 36(5), pp. 9498-9505, 2009.

[20] C. W. Lin, T. P. Hong, and W. H. Lu, "Efficiently mining high average utility itemsets with a tree structure," *International Conference on Intelligent Information and*

Database Systems, pp. 131-139, 2010.

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

[22] G. C. Lan, T. P. Hong, and V. S. Tseng, "Efficient 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.

[23] C. W. Lin, G. C. Lan, and T. P. Hong, "An incremental mining algorithm for high utility itemsets," *Expert Systems with Applications*, vol. 39(8), pp. 7173-7180, 2012.

[24] M. Y. Lin, T. F. Tu, and S. C. Hsueh, "High utility pattern mining using the maximal itemset property and lexicographic tree structures," *Information Sciences*, vol. 215, pp. 1-14, 2012.

[25] 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.

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

[27] 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.

[28] 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. 13, pp. 100-115, 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] R. Martinez, N. Pasquier, and C. Pasquier, "GenMiner: Mining non-redundant association rules from integrated gene expression data and annotations," *Bioinformatics*, vol. 24, no. 22, pp. 2643-2644, 2008.

[31] 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.

[32] 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.

[33] H. Yao, H. J. Hamilton, and L. Geng, "A unified framework for utility based measures for mining itemsets," *International Workshop on Utility-Based Data Mining*, pp. 482-486, 2004.

[34] C. H. Yun and M. S. Chen, "Using pattern-join and purchase-combination for mining web transaction patterns in an electronic commerce environment," *Computer Software and Applications Conference*, pp. 99-104, 2000.

Jerry Chun-Wei Lin is currently working as an Associate Professor at School of Computer Science and Technology, Harbin Institute of Technology Shenzhen Graduate School, Shenzhen, China. He has published more than 200 research papers in referred journals and international conferences. His research interests include data mining, soft computing, artificial intelligence, social computing, multimedia and image processing, and privacy-preserving and security technologies. He is also the project leader of SPMF: An Open-Source Data Mining Library, which is a toolkit offering multiple types of data mining algorithms. He also serves as the Editor-in-Chief of the international journal of Data Science and Pattern Recognition (DSPR).

Shifeng Ren is currently a master student at School of Computer Science and Technology, Harbin Institute of Technology Shenzhen Graduate School, Shenzhen, China. His research interests include data mining, pattern discovery, and big data technologies.

Philippe Fournier-Viger is a full professor and Youth 1000 scholar at the Harbin Institute of Technology Shenzhen Grad. School, China. He received a Ph.D. in Computer Science at the University of Quebec in Montreal (2010). He has published more than 150 research papers in refereed international conferences and journals, which have received more than 1,400 citations. His research interests include data mining, pattern mining, sequence analysis and prediction, text mining, e-learning, and social network mining. He is the founder of the popular SPMF open-source data mining library, which has been cited in more than 450 research papers since 2010. He is also editor-in-chief of the Data Mining and Pattern Recognition journal.

Tzung-Pei Hong received his B.S. degree in chemical engineering from National Taiwan University in 1985, and his Ph.D. degree in computer science and information engineering from National Chiao-Tung University in 1992. He served at the Department of Computer Science in Chung-Hua Polytechnic Institute from 1992 to 1994, and at the Department of Information Management in I-Shou University from 1994 to 2001. He was in charge of the whole computerization and library planning for National University of Kaohsiung in Preparation from 1997 to 2000 and served as the first director of the library and computer center in National University of Kaohsiung from 2000 to 2001, as the Dean of Academic Affairs from 2003 to 2006, as the Administrative Vice President from 2007 to 2008, and as the Academic Vice President in 2010. He is currently a distinguished professor at the Department of Computer Science and Information Engineering and at the Department of Electrical Engineering, National University of Kaohsiung, and a joint professor at the Department of Computer Science and Engineering, National Sun Yat-sen University, Taiwan. He got the first national flexible wage award from Ministry of Education in Taiwan. He has published more than 500 research papers in international/national journals and conferences and has planned more than fifty information systems. He is also the board member of more than forty journals and the program committee member of more than five hundred conferences. His current research interests include knowledge engineering, data mining, soft computing, management information systems, and WWW applications.