



Mining High-Average Utility Itemsets with Positive and Negative External Utilities

Irfan Yildirim¹ · Mete Celik¹

Received: 27 March 2019 / Accepted: 28 October 2019 / Published online: 7 November 2019
© Ohmsha, Ltd. and Springer Japan KK, part of Springer Nature 2019

Abstract

High-utility itemset mining (HUIM) is an emerging data mining topic. It aims to find the high-utility itemsets by considering both the internal (i.e., quantity) and external (i.e., profit) utilities of items. High-average-utility itemset mining (HAUIM) is an extension of the HUIM, which provides a more fair measurement named average-utility, by taking into account the length of itemsets in addition to their utilities. In the literature, several algorithms have been introduced for mining high-average-utility itemsets (HAUIs). However, these algorithms assume that databases contain only positive utilities. For some real-world applications, on the other hand, databases may also contain negative utilities. In such databases, the proposed algorithms for HAUIM may not discover the complete set of HAUIs since they are designed for only positive utilities. In this study, to discover the correct and complete set of HAUIs with both positive and negative utilities, an algorithm named MHAUIPNU (mining high-average-utility itemsets with positive and negative utilities) is proposed. MHAUIPNU introduces an upper bound model, three pruning strategies, and a data structure. Experimental results show that MHAUIPNU is very efficient in reducing the size of the search space and thus in mining HAUIs with negative utilities.

Keywords High-average-utility itemset mining · Negative utility · Utility mining · Data mining

✉ Irfan Yildirim
irfanyildirim@erciyes.edu.tr

Mete Celik
mcelik@erciyes.edu.tr

¹ Department of Computer Engineering, Erciyes University, 38039 Kayseri, Turkey

Introduction

Frequent itemset mining (FIM) [1, 3, 6, 8, 15], which is one of the well-known research topics in data mining, is used to discover frequently occurred itemsets from databases. However, the frequency of itemsets may not be meaningful or sufficient for different real-world mining problems. For example, frequently purchased items of a retail market are likely to be low-profitable products (items) since cheap products tend to be sold more. In addition, infrequent itemsets may be highly profitable. Because of these reasons, FIM is not effective to determine which itemsets are highly profitable or which itemsets have strong impacts on increasing the market profit of a retail, etc. FIM considers only binary information of items and assumes that each item has the same importance.

To overcome these limitations, high-utility itemset mining (HUIM) [10, 22–24, 30, 31] has been studied. HUIM considers both internal and external utilities of items instead of their frequencies on discovering itemsets. In a quantitative transactional database of a retail, the purchased quantity of an item in transactions can be evaluated as its internal utilities and the profit obtained when one unit of it is sold can be considered as its external utility. However, HUIM is a challenging problem since the utility of an itemset is neither monotonic nor anti-monotonic. That is, a superset or a subset of an itemset may have lower or higher utility than the itemset has. Therefore, studies for HUIM have been mainly focused on introducing tighter upper bounds on the utility values of itemsets to reduce the search space by providing an anti-monotonic property.

However, when the length of an itemset is increased, the utility of the itemset tends to be greater. The reason is that the total utility of an itemset is the sum of utility of each item that the itemset includes. Thus, HUIM suffers from a large number of high-utility itemset (HUI) generation and inclusion of low-profit item(s) in these HUIs.

To address this issue, high-average-utility itemset mining (HAUIM) is introduced with a more fair utility measurement called average-utility (au) [7]. The au value of an itemset is obtained by dividing its utility by the number of items that it has. If an itemset has an au value which is not lower than a pre-defined utility threshold ($minUtil$) then it is called a high-average-utility itemset (HAUI).

The problem of HAUIM is challenging similar to the problem of HUIM since the au measurement is neither monotonic nor anti-monotonic as the utility measurement of HUIM. Thus, au values of itemsets cannot be directly used to reduce the search space, and so an upper bound measure named average-utility upper bound ($auub$) is used to prune unpromising itemsets early. The $auub$ is introduced by TPAU algorithm [7] which is a level-wise approach. TPAU algorithm mines HAUIs in two phases. It generates a complete set of candidate HAUIs utilizing their $auub$ values in the first phase via a database scan. Then it calculates actual au values of them to determine HAUIs in the second phase via another database scan. Since then, several approaches, such as PBAU [13], PAI [12], HAUP-Growth [14], HAUI-Tree [25], HAUI-Miner [18], EHAUPM [19], MHAI [35], FHAUIM [20], TUB-HAUPM [32], and dHAUIM [29], have been introduced

with their own data structures, upper bounds, and optimization techniques to increase the efficiency of HAUIM.

However, the HAUIM algorithms proposed in the literature assume that items have only positive external utilities. On the other hand, databases may also contain negative external utilities in many real-world applications as mentioned in [2, 11, 17, 28]. For example, large supermarket chains, such as Walmart, run hundreds or thousands of bundled promotion or cross-promoted campaigns where many products are offered with at a loss (negative utility) [11]. By selling products with negative margins (or utilities), the supermarkets aim to increase their overall profits by selling related (cross-promoted) products. Therefore, from a business perspective, mining out meaningful (or valuable) sets of products (or itemsets) from the past data is very important for understanding purchasing behaviors of customers, and thereby developing new sale campaigns that will increase future profitability. However, when databases also contain negative utilities, existing HAUIM algorithms may face the problem of incomplete discovery of HAUIs. Consequently, developing an approach that preserves the completeness and correctness of the set of discovered HAUIs in a database having negative utilities is important.

This study deals with developing techniques for efficient mining of HAUIs in databases that include positive and negative utilities. The main contributions of this study are as follows:

- An algorithm, named as MHAUIPNU (Mining High-Average-Utility Itemsets with Positive and Negative Utilities), is proposed to mine HAUIs out of databases having both positive and negative utilities.
- A new upper bound model called *tubpn* (tighter upper bound with positive and negative utilities) is proposed to reduce the search space for mining HAUIs with positive and negative utilities.
- A list data structure is developed to store required information for HAUIM to reduce the number of database scans.
- Three novel pruning strategies are designed to speed up the performance of the proposed algorithm. The first pruning strategy is designed based on the proposed *tubpn* model. Others are designed by utilizing the properties associated with items (or itemsets) having negative utilities.
- Experiments are conducted on different datasets to show the efficiency of proposed MHAUIPNU algorithm.

The rest of the paper is organized as follows. Discussion of the related work is given in the next section. The following section gives the basic concepts of the problem of HAUIM. The proposed algorithm is described in the subsequent section. Before the concluding section, experimental analyses are given. The final section presents conclusions and future works.

Related Work

In the literature, there are several HUIM and HAUIM studies solving the problem of HAUIM with only positive utilities. However, there is a limited number of HUIM studies [2, 11, 17, 28] designed for positive and negative utilities and, to the best of our knowledge, there is no HAUIM study solving the problem of HAUIM with positive and negative utilities.

Based on algorithmic structures, the studies of HUIM and HAUIM can be classified as two-phased and single-phased algorithms. Two-phased HUIM (or HAUIM) algorithms are designed to discover a set of itemsets that are potentially HUIs (or HAUIs) in the first phase. Then, in the second phase, they calculate the actual values of the discovered itemsets to decide which ones are HUIs (or HAUIs) by performing an additional database scan. On the other hand, single-phased HUIM (or HAUIM) algorithms are designed to discover the HUIs (or HAUIs) by directly obtaining the actual values of the itemsets from the data structures they store the necessary information for mining.

This study deals with developing a single-phased algorithm to discover HAUIs out of databases having positive and negative utilities. The literature review of HUIM and HAUIM studies is given in the following subsections, respectively.

High-Utility Itemset Mining

Two-phase [24] algorithm is first presented to solve the problem of HUIM by introducing the concept of TWU model for the purpose of using downward-closure property of candidate itemsets to prune the search space of HUIM. It uses an iterative level-wise approach by employing a generate-and-test strategy. However, it suffers from multiple database scans and generation of many candidate itemsets. In time, tree-based algorithms such as HUP-Tree [16], UP-Growth [31], and UP-Growth+ [30] were introduced to solve the HUIM problem more efficiently. These algorithms operate in two phases. In addition to TWU model, some of these algorithms use additional strategies to prune the search space more effectively. For example, UP-Growth+ prunes candidate itemsets more effectively using four different strategies [30]. All these tree-based approaches are more effective than Two-phase algorithm since they reduce the number of database scans. However, they still suffer from the drawbacks of performing the mining task based on two phases.

To avoid the limitation of the two-phased algorithms, single-phased algorithms including HUI-Miner [23], FHM [5], HUP-Miner [10], d2HUP [22], EFIM [37], IMHUP [27], and mHUIMiner [26] were proposed. These algorithms are more efficient than previous algorithms since they mine HUIs without a candidate generation phase. They use different optimization strategies to prune the search space of candidates for improving the performance of mining HUIs. Among them, d2HUP [22] uses a data structure based on hyper-link while the others use list-based data structures.

The problem of mining HUIs with positive and negative utilities was first solved by HUINIV-Mine [2] algorithm. The HUINIV-Mine uses a re-defined TWU model to discover HUIs without missing any HUIs. However, since it is an extended form of Two-phase [24], it has same major drawbacks, such as generating a large number of HUIs and scanning the database multiple times. Then FHN [17] algorithm, which is an extended form of FHM [5], was introduced with a list structure named PNU list (Positive and Negative Utility List). The FHN is a single-phase algorithm, and so it is much faster than the HUINIV-Mine. However, the FHN still requires much time and memory to execute its mining task. To increase the performance of the problem of HUIM with negative unit profits, more efficient approaches including GHUM [11] and EHIN [28] were also proposed. The GHUM uses a simplified utility-list structure to keep required information for mining and utilizes four different strategies which are named as generalized U-Prune, LA-Prune, A-Prune, and N-Prune [11]. The EHIN algorithm is an extended version of EFIM [28]. It uses a projection technique and merges identical transactions to reduce the cost of database scans. It uses two pruning strategies, named as redefined subtree utility and local utility, to prune the search space using a depth-first search.

High-Average-Utility Itemset Mining (HAUIM)

In the literature, several HAUIM algorithms have been presented. The first algorithm is TPAU which is a two-phased algorithm [7]. To reduce the search space, it uses an upper bound called the average-utility upper bound (*auub*) on utilities of itemsets to obtain the downward-closure property. The downward-closure property of *auub* ensures that if an itemset is not a HAU based on its *auub* value then none of its supersets (extensions) can be a HAU because of the fact that *auub* value of an itemset cannot be lower than *auub* value of any of its supersets [7]. To speed up to task of mining HAUIs, projection-based algorithms such as PBAU [13] and PAI [12] were introduced. However, these algorithms mainly suffer from multiple database scans and generation of a large number of candidate itemsets. To overcome multiple database scans and reduce the number of candidates, tree-based algorithms, such as HAUP-Growth [14] algorithm and HAUL-Tree [25] algorithm were presented.

Then several single-phased algorithms were proposed, such as HAUI-Miner [18], EHAUPM [19], MHAI[35], FHAUPM [20], TUB-HAUPM [32], and dHAUIM [29] algorithms. To improve the performance of mining HAUIs further, these algorithms introduced their own upper bounds, pruning strategies, and data structures. Among them, dHAUIM [29] uses a horizontal representation of the database, unlike the others. Besides, TUB-HAUPM [32] does not have a data structure. The remaining algorithms use list-based data structures to store the required information for discovering HAUIs.

In addition, several variants of the problem of HAUIM have also been studied, such as efficiently updating of HAUIs with transaction insertion [33] and deletion [21], incremental HAUIM [9, 34], and HAUIM over data stream with the damped window technique [36].

Table 1 A sample transactional database, *DB*

TID	Transaction
T_1	($a : 5$) ($b : 2$) ($c : 1$) ($d : 2$)
T_2	($a : 1$) ($c : 1$) ($d : 1$) ($g : 3$)
T_3	($a : 1$) ($c : 1$) ($f : 1$)
T_4	($a : 1$) ($f : 4$) ($g : 2$)
T_5	($a : 1$) ($g : 2$)
T_6	($b : 3$) ($c : 2$) ($d : 3$) ($e : 1$)
T_7	($c : 6$) ($e : 4$)
T_8	($e : 1$) ($f : 3$)

Table 2 External utility table of *DB*, *EUT(DB)*

Item	a	b	c	d	e	f	g
External utility	3	6	-3	12	-5	-2	-1

However, all these HAUIM studies were designed by assuming that databases contain only positive utilities. In contrast to these studies, this study deals with developing techniques for efficient mining of the complete and correct set of HAUIs from a database containing both positive and negative utilities.

Basic Concepts and Problem Statement

In this section, definitions and key notations related to the problem of HAUIM are given.

Let the set of items I consist of n distinct items, $I = \{i_1, i_2, \dots, i_n\}$ and each item $i \in I$ has an external utility $eu(i)$, e.g., its unit profit. A transaction T_j can include any group (or set) of items from I , $T \subseteq I$, where j is the identification of T_j called its *TID*. Each item i in each T_j , where $i \in I$, has a positive number called as its internal utility $iu(i)$, e.g., its purchase quantity. A transactional database *DB* comprises a set of transactions $DB = \{T_1, T_2, \dots, T_m\}$, where m is the total number of transactions in the *DB*.

Let us consider the sample *DB* given in Table 1. The *DB* consists of eight transactions and the set of items $I = \{a, b, c, d, e, f, g\}$. Let us take T_1 as an example. In T_1 , four items which are a, b, c, d exist with their internal utilities which are 5, 2, 1, and 2, respectively. External utilities of each items of the sample *DB* are given in Table 2. External utility of item i can be either positive or negative. As can be seen in Table 2, items c, e, f , and g have negative external utilities while others have positive external utilities.

Note that Tables 1 and 2 will be used in the running examples given in the rest of the paper. Besides, “positive items” (*PIs*) and “negative items” (*NIs*) terms will be used to denote, respectively, items having positive and negative external utilities. For the running example, $PIs = \{a, b, d\}$ and $NIs = \{c, e, f, g\}$.

Definition 1 (*Utility of an item in a transaction [7]*) The utility of an item i in a transaction T_j is denoted as $u(i, T_j)$ and calculated as multiplication of its internal utility in the transaction T_j $iu(i, T_j)$ and its external utility $eu(i)$:

$$u(i, T_j) = iu(i, T_j) \times eu(i). \tag{1}$$

For example, $iu(b, T_1)$ is obtained as 2 from the Table 1 and $eu(b)$ is obtained as 6 from the Table 2. Therefore, $u(b, T_1) = 2 \times 6 = 12$.

Definition 2 (*Utility of an itemset in a transaction [7]*) The utility of an itemset X in a transaction T_j is denoted as $u(X, T_j)$ and calculated by summing $u(i, T_j)$ value of each item i , where $\forall i \in X \subseteq T_j$:

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

For example, the utility of itemset $X = \{b, d\}$ in transaction T_1 is calculated as $u(X, T_1) = u(b, T_1) + u(d, T_1) = 12 + 24 = 36$.

Definition 3 (*Utility of an itemset in a database [7]*) The utility of an itemset X in a database is denoted as $u(X)$ and calculated by summing each $u(X, T_j)$ value, where $\forall T_j \supseteq X$.

$$u(X) = \sum_{X \subseteq T_j} u(X, T_j). \tag{3}$$

For example, consider that itemset $X = \{b, d\}$. Since X is found in T_1 and T_6 , $u(X)$ is calculated as $u(\{b, d\}, T_1) + u(\{b, d\}, T_6) = 36 + 54 = 90$.

Definition 4 (*Average-utility of an itemset in a database [7]*) The average-utility of an itemset X in a database is denoted as $au(X)$ and calculated by dividing $u(X)$ by the length of X .

$$au(X) = u(X)/|X|. \tag{4}$$

For example, since $u(\{b, d\}) = 90$ and $|\{b, d\}| = 2$, $au(\{b, d\}) = 90/2 = 45$.

Definition 5 (*High-average-utility itemset [7]*) If an itemset X has an average-utility value which is not less than a predefined minimum utility threshold $minUtil$, then it is called as a high-average-utility itemset (HAUI).

$$HAUI \leftarrow \{X | au(X) \geq minUtil\}. \tag{5}$$

For example, if $minUtil$ is 15, then itemset $\{b, d\}$ is a HAUI since $au(\{b, d\}) = 45 \geq minUtil = 15$.

The average-utility of an itemset does not satisfy the downward-closure property. Therefore, the au values of itemsets cannot be used to prune the search space. To address this problem, the average-utility upper bound ($auub$) model was introduced by TPAU algorithm [7] to obtain a downward-closure property by overestimating

the average-utility values of itemsets. The *auub* model was also used by the other algorithms [12–14, 18–20, 25, 32, 35] to determine the set of promising items.

Definition 6 (The *auub* value of an itemset [7]) Let transaction maximum utility (*tmu*) of a transaction T_j be denoted as $tmu(T_j)$, where $tmu(T_j) = \max\{u(i_k, T_j) \mid i_k \in T_j\}$. The *auub* of an itemset X in a database is denoted as $auub(X)$ and calculated by summing each $tmu(T_j)$ value obtained from transactions that contain the itemset X :

$$auub(X) = \sum_{X \subseteq T_j} tmu(T_j). \quad (6)$$

For example, items b and d appear together in transactions T_1 and T_6 . The values $tmu(T_1)$ and $tmu(T_6)$ are calculated as $\max(u(a, T_1), u(b, T_1), u(c, T_1), u(d, T_1)) = \max(15, 12, -3, 24) = 24$ and $\max(u(b, T_6), u(c, T_6), u(d, T_6), u(e, T_6)) = \max(18, -6, 36, -5) = 36$, respectively. Thus, $auub(\{b, d\})$ is calculated as $tmu(T_1) + tmu(T_6) = 24 + 36 = 60$.

Definition 7 (High average-utility upper-bound itemset [7]) If an itemset X has an *auub* value which is not less than the *minUtil*, then it is called as a high-average-utility upper bound itemset (HAUUBI):

$$HAUUBI \leftarrow \{X \mid auub(X) \geq minUtil\}. \quad (7)$$

For example, itemset $\{b, d\}$ is a HAUUBI since $auub(\{b, d\}) = 60 \geq minUtil = 15$. Note that, if the length of an HAUUBI is k then it is called as a k -HAUUBI. Since the length of $\{b, d\}$ is 2, $\{b, d\}$ is a 2-HAUUBI.

Property 1 (Pruning the search space by utilizing the *auub* [7]) *HAUIs* \subseteq *HAUUBIs* is true based on the downward-closure property of *auub* model. Thus, if an itemset X is not a HAUUBI, then it is not a HAUI and none of its extensions (supersets) can be a HAUI. Therefore, X can be pruned from the search space.

Discussion of auub model with negative utilities: The *auub* model is designed based on the assumption that all items in the database have positive utilities. Therefore, using *auub* model with negative utilities may result in missing some HAUIs since *tmu* values of transactions become negative when all items they include have negative external utilities (e.g., $tmu(T_7) = -18$). Therefore, such transactions reduce the *auub* values of the negative items and may cause some negative items to have *auub* lower than *minUtil*. Based on Property 1, if an item has *auub* lower than *minUtil*, then it is pruned. Thus, *auub* model may cause to miss some HAUIs with negative utilities. As a result, the downward-closure property of *auub* does not hold for a set of negative items and their extensions.

For example, let us take item e as an example. Since item e appears in T_6 , T_7 , and T_8 , *auub* value of e is $auub(e) = tmu(T_6) + tmu(T_7) + tmu(T_8) = 36 + (-18) + (-5) = 13 \not\geq minUtil = 15$. Thus, item e will be pruned by the *auub* model and its extensions will not be examined. However, itemset $\{b, d, e\}$ which is an extension (superset) of item e is a HAUI since b , d , and e occurred together in T_6 and $au(\{b, d, e\})$

is obtained as $16.3 > \min Util = 15$. As can be seen, $auub(e) = 13 < au(\{b, d, e\}) = 16.3 < auub(\{b, d, e\}) = 32$, the downward-closure property of $auub$ may not hold between a negative item and its extensions.

For the correctness and completeness of the HAUIM in databases with negative utilities, the downward-closure property of $auub$ can be held by ignoring transactions containing only negative items. For this reason, Definition 8 revises the $auub$ model as $auub^{pn}$ ($auub$ with positive and negative utilities) model.

Definition 8 (*The $auub^{pn}$ of an itemset*) The $auub$ with positive and negative utilities of an itemset X is denoted as $auub^{pn}(X)$ and defined as follows:

$$auub^{pn}(X) = \sum_{X \subseteq T_j \wedge tmu(T_j) > 0} tmu(T_j). \tag{8}$$

Mining High-Average-Utility Itemsets with Both Positive and Negative External Utilities

This paper proposes an algorithm for mining HAUIs out of a database containing positive and negative utilities with a novel upper model, three pruning strategies, and a list data structure. The proposed algorithm is named as MHAUIPNU (Mining High-Average-Utility Itemset with Positive and Negative Utilities).

The next subsection introduces the proposed upper bound model named $tubpn$ (tighter upper bound with positive and negative utilities) followed by which the pruning strategies that MHAUIPNU utilizes are discussed. The subsequent subsection presents a list structure called tighter upper bound with positive and negative utilities (TUBPNU) list to store required information for mining HAUIs. The final subsection introduces the MHAUIPNU algorithm and then gives the execution trace of the MHAUIPNU algorithm on the running example.

Proposed Tighter Upper Bound with Positive and Negative Utilities ($tubpn$)

Since the $auub^{pn}$ does not take into account transactions having negative utilities, it can be used to solve the problem of HAUIM with positive and negative utilities. However, since it is a modified version of $auub$ model by ignoring transaction having negative utilities, it is a loose upper bound as the $auub$.

This paper proposes a novel tighter upper bound model named as $tubpn$ (tighter upper bound with positive and negative utilities). The following definitions are associated with the $tubpn$ model, considering that $fmu(T_j)$ and $smu(T_j)$ denote transaction first and second maximum utilities in a transaction T_j , respectively, where $|T_j| > 1$. If $|T_j| = 1$, then $smu(T_j) = 0$.

Definition 9 (*Tighter upper bound of a positive item in a transaction*) The tighter upper bound of a positive item p in a transaction T_j , where $p \in T_j$, is denoted as $tubp(p, T_j)$ and defined as follows:

$$tubp(p, T_j) = \begin{cases} fmu(T_j), & \text{if } u(p, T_j) = fmu(T_j) \\ \frac{fmu(T_j) + smu(T_j)}{2}, & \text{otherwise.} \end{cases} \quad (9)$$

For example, $u(b, T_1)$, $u(d, T_1)$, $fmu(T_1)$, and $smu(T_1)$ are 12, 24, 24, and 15, respectively. Besides, $u(b, T_1) \neq fmu(T_1) = u(d, T_1)$. Therefore, $tubp(b, T_1) = 19.5 (= (24 + 15)/2)$ and $tubp(d, T_1) = 24$.

Definition 10 (*Tighter upper bound of a negative item in a transaction*) Let T_j .|PIs| be the number of positive items in a transaction T_j . The tighter upper bound of a negative item n in a transaction T_j , where $n \in T_j$, is denoted as $tubn(n, T_j)$ and defined as follows:

$$tubn(n, T_j) = \begin{cases} \frac{fmu(T_j) + smu(T_j)}{2}, & \text{if } T_j.\text{|PIs|} > 1 \\ \frac{fmu(T_j) + u(n, T_j)}{2}, & \text{else if } fmu(T_j) + u(n, T_j) > 0 \\ 0, & \text{otherwise.} \end{cases} \quad (10)$$

For example, $tubn(e, T_6) = (fmu(T_6) + smu(T_6))/2 = (36 + 18)/2 = 27$ since $T_6.\text{|PIs|} > 1$. For another example, $tubn(f, T_4) = 0$ since $T_4.\text{|PIs|} = 1$ and $fmu(T_4) + u(f, T_4) = 3 + (-8) \not> 0$.

Definition 11 (*Tighter upper bound of an itemset in a transaction*) Let X be an itemset considering $X.\text{PIs}$ and $X.\text{NIs}$ is the set of positive and negative items that X contains, respectively. The tighter upper bound of X in a transaction T_j , where $X \subseteq T_j$, is denoted as $tubpn(X, T_j)$ and defined as follows:

$$tubpn(X, T_j) = \begin{cases} \min\{tubn(n_k, T_j) | n_k \in X.\text{NIs}\}, & \text{if } X.\text{NIs} \neq \emptyset, \\ \min\{tubp(p_k, T_j) | p_k \in X.\text{PIs}\}, & \text{otherwise.} \end{cases} \quad (11)$$

For example, for the itemset $X = \{b, d\}$, $tubpn(X, T_1)$ is calculated as $\min\{tubp(b, T_1), tubp(d, T_1)\} = \min\{19.5, 24\} = 19.5$. For another example, $tubpn(\{a, f, g\}, T_4)$ is obtained as follows. Since $\{a, f, g\}.\text{NIs} \neq \emptyset$, $tubpn(\{a, f, g\}, T_4) = \min\{tubn(f, T_4), tubn(g, T_4)\} = \min\{0, 0.5\} = 0$.

Definition 12 (*Tighter upper bound of an itemset in a database*) The tighter upper bound of an itemset X in a database DB is denoted as $tubpn(X)$ and defined as follows:

$$tubpn(X) = \sum_{X \subseteq T_j} tubpn(X, T_j). \quad (12)$$

For example, for the itemset $X = \{b, d\}$, $tubpn(X)$ is calculated as $tubpn(X, T_1) + tubpn(X, T_6) = 19.5 + 27 = 46.5$.

Theorem 1 (In a database, $tubpn(X) \geq au(X)$ is correct for any itemset X) In a database, the average-utility value of any itemset X cannot be greater than its $tubpn$ value, that is, $tubpn(X) \geq au(X)$.

Proof Let X and $X.Tids$ be an itemset and a set of transactions containing X in a database, respectively. The proof of $tubpn(X) \geq au(X)$ can be provided by presenting $tubpn(X, T_j) \geq au(X, T_j)$ is correct for each transaction $T_j \in X.Tids$, where $au(X, T_j) = u(X, T_j) / |X|$. Obviously, $au(X) = u(X) / |X| = \sum_{T_j \in X.Tids} au(X, T_j)$ is correct. However, X can consist of either only positive items (Case1) or negative items (Case2) or both of positive and negative items (Case3). Therefore, for all three cases, $tubpn(X, T_j) \geq au(X, T_j)$ is correct as presented below.

- Case 1 Let itemset X and T_j be any set of items that consists only positive items and a transaction, respectively, such that $X \in T_j$.
 - If $|X| = 1$ and $fmu(T_j) = u(X, T_j)$, then $tubpn(X, T_j) = fmu(T_j) = u(X, T_j) = au(X, T_j)$ is clear.
 - If $|X| = 1$ and $fmu(T_j) \neq u(X, T_j)$, then $tubpn(X, T_j) = (fmu(T_j) + smu(T_j)) / 2 > u(X, T_j) = au(X, T_j)$ is clear.
 - If $|X| > 1$, then $tubpn(X, T_j) = \min\{tubp(p_k, T_j) | p_k \in X.PIs\} = (fmu(T_j) + smu(T_j)) / 2 \geq u(X, T_j) / |X| = au(X, T_j)$ is clear.
- Case 2 Let itemset X and T_j be any set of items that consists only negative items and a transaction, respectively, such that $X \in T_j$. Therefore, $tubpn(X, T_j) = \min\{tubn(n_k, T_j) | n_k \in X.NIs\} = 0$ since $tubn(n_j, T_j) = 0$ for each $n_j \in T_j$. Besides, $u(X, T_j) < 0$ is clear. Then $tubpn(X, T_j) > au(X, T_j)$ is clear.
- Case 3 Let itemset X and T_j be any set of items that consists both positive and negative items and a transaction, respectively, such that $X \in T_j$. Thus, $tubpn(X, T_j)$ is calculated as $\min\{tubn(n_k, T_j) | n_k \in X.NIs\}$. The length of $X.PIs$ can be greater than or equal to 1.
 - If $|X.PIs| > 1$, then $tubpn(X, T_j) = (fmu(T_j) + smu(T_j)) / 2 > u(X, T_j) / |X| = au(X, T_j)$ is clear.
 - If $|X.PIs| = 1$, then $tubpn(X, T_j) = (fmu(T_j) + \min\{u(n_k, T_j) | n_k \in X.NIs\}) / 2 \geq u(X, T_j) / |X| = au(X, T_j)$ is clear. □

Property 2 (The downward-closure property of $tubpn$ model) Let X be an itemset and Y be any superset of X , such that $X \subseteq Y$. Therefore, $tubpn(X) \geq tubpn(Y)$ holds.

Proof Let X be an itemset and Y be any superset of X , such that $X \subset Y$. Considering $X.Tids$ and $Y.Tids$ be two sets of transactions containing X and Y , respectively, the downward-closure property of $tubpn$ is explained as follows. Since $X.Tids \supseteq Y.Tids$ is true, then $tubpn(X) \geq tubpn(Y)$ is also true. As a result, $tubpn(X)$ value of any X is

anti-monotonic. Therefore, *tubpn* model satisfies the downward-closure property. \square

Property 3 (The *tubpn* model is tighter than the *auub^{pn}* or *auub* model) The *tubpn*(X) and *auub^{pn}*(X) (or *auub*(X)) values for any itemset X are equal if X is a positive item and $u(X, T_j) = fmu(T_j)$ is correct for each transaction T_j , such that $X \in T_j$. In all other cases, the *tubpn*(X) < *auub^{pn}*(X) (or *auub*(X)).

Proof The $fmu(T_j)$ (= $tmu(T_j)$) value is considered only in the calculation of the *tubpn*(p, T_j), where p is a positive item and $p \in T_j$. Thus, for *tubpn*(X) and *auub^{pn}*(X) to be equal, the itemset X must contain only a positive item and $u(X, T_j) = fmu(T_j)$ must correct for all transactions which contain X . Otherwise (if $X = n$ where item $n \in NIs$ or $|X| > 1$), *tubpn*(X) < *auub^{pn}*(X) holds since *tubpn*(X, T_j) is always lower than the $fmu(T_j)$, that is, the *tubpn* model is tighter than the *auub^{pn}* (or *auub*(X)) model, and thus Property 3 is correct. \square

Pruning Strategies

In this subsection, the proposed three pruning strategies of the MHAUIPNU algorithm are introduced. The first pruning strategy is based on the proposed *tubpn* model. The second and third pruning strategies are designed by utilizing the properties associated with items having negative utilities.

Pruning Strategy 1 (Pruning the search space by utilizing *tubpn* model) Let X be an itemset. If *tubpn*(X) < *minUtil*, then X can be directly pruned from the search space.

Proof The correctness of *tubpn*(X) \geq *tubpn*(Y) for the itemsets X and any Y , such that $X \subset Y$, is explained by Property 2. Thus, itemsets with a lower *tubpn* value than the *minUtil* and the extensions of these itemsets cannot be HAUIs. \square

The *tubpn* model can be used to ensure the completeness and correctness of the MHAUIPNU algorithm for mining HAUIs in a database with positive and negative utilities. However, the MHAUIPNU utilizes two additional pruning strategies to reduce the size of the search space of the problem more efficiently. These strategies are designed based on some properties associated with itemsets with negative utilities. Details are given below.

Lemma 1 A HAUI should contain at least one positive item.

Rationale For any itemset X having only negative items, $u(X) < 0$ is obviously correct. Thus, an itemset having only negative items cannot be a HAUI. \square

Based on Lemma 1, the second pruning strategy is designed as follows.

Pruning Strategy 2 (*Pruning the negative itemsets*) Let n be a negative item. Then n and none of its negative extensions can be a HAUI. Itemsets having only negative items can be directly pruned.

The following property is related to the last pruning strategy.

Property 4 (*Relationship between positive utilities of an itemset and its negative extensions*) Let X, N , and XN be an itemset, any set of negative items, and a negative extension of X , such that $XN = \{X \cup N\}$, respectively. Besides, let $X.PU$ and $X.NU$ be the sum of positive utilities and negative utilities of an itemset X , respectively. Thus, $u(X) = X.PU + X.NU$. The relationship between positive utility values of X and XN is that $X.PU \geq XN.PU$

Proof Consider that $X.PIs$ is the set of positive items that X contains. Since $X.Tids \geq XN.Tids$ and $X.PIs = XN.PIs$, then $X.PU \geq XN.PU$ is clear. □

Pruning Strategy 3 (*Pruning the negative extensions of an itemset*) If $X.PU / (|X| + 1) < minUtil$, then none of negative extensions of the itemset X can be a HAUI.

Proof Based on Property 4 and its proof, we know that $X.PU \geq XN.PU$ holds between an itemset X and any of its negative extensions XN . Besides, $|X| + 1 \leq |XM|$ is correct. Moreover, since $u(XN) = XN.PU + XN.NU$, $X.PU > u(XN)$ is also correct. Therefore, $X.PU / (|X| + 1) > au(XN) = u(XN) / |XM|$ is true. □

However, a question arises with Pruning Strategy 3, that is, how to calculate the positive utility of a negative extension of an itemset. The following definition and property are associated with this question.

Definition 13 (*Positive utility array of an itemset*) The positive utility array (PUA) of an itemset X , denoted as $PUA(X)$, is a one-dimensional list. $PUA(X)[j]$ stores the positive utility of X obtained from transaction T_j , such that $X \subseteq T_j$, and $PUA(X)[j]$ is calculated as follows:

$$PUA(X)[j] = \sum_{p_k \in X.PIs \wedge X \subseteq T_j} u(p_k, T_j). \tag{13}$$

Property 5 (*Calculating the positive utility of a negative extension of an itemset*) Let X be an itemset and XN be any negative extension of X . $XN.PU$ can be obtained by summing each $PUA(X)[j]$ value, such that $\forall j \in XN.Tids$.

Proof Property 5 is clear since $XN.Tids \subseteq X.Tids$, and $PUA(X)[j] = PUA(XN)[j]$ is correct for each $j \in XN.Tids$. □

Proposed Tighter Upper Bound with Positive and Negative Utilities (TUBPNU) List Structure

To avoid multiple database scans and prune the search space of HAUIs with positive and negative utilities by utilizing the proposed *tubpn* model and three pruning strategies, a list data structure is proposed. It is named as tighter upper bound with positive and negative utilities (TUBPNU) list.

Definition 14 (*TUBPNU list of an itemset*) The TUBPNU list of an itemset X is denoted as $TUBPNUL(\{X\})$. It consists of a set of elements. Each element E in $TUBPNUL(\{X\})$ consists of three fields (*tid*, *utility*, *tubpn*) for each transaction T_j (see Fig. 2), such that $X \subseteq T_j$ and $PIs \cap T_j \neq \emptyset$. *tid* is the transaction identification of T_j . *utility* and *tubpn* are the values that represent $u(X, T_j)$ and $tubpn(X, T_j)$, respectively. Note that, *tubpn* values are rounded up and stored as integers in the TUBPNU lists to reduce the memory consumption of the proposed TUBPNU list data structure.

For example, $tubpn(\{a\}, T_1) = 19.5$. When it is rounded up, it becomes 20. Thus, the element in $TUBPNUL(\{a\})$ for T_1 is stored as (1, 15, 20).

Definition 15 (*Total processing order < in MHAUIPNU algorithm*) The search space of the problem can be represented as an enumeration tree. The enumeration tree of a set of items $I = \{i_1, i_2, \dots, i_n\}$ is a tree-shaped search structure constructed by enumerating 2^n itemsets (subsets of I) based on a total processing order (see Fig. 1). In the previous studies, it was shown that selecting a proper processing order can decrease the size of the enumeration tree. The MHAUIPNU algorithm utilizes the properties associated with negative items to reduce the size of the search space (by Pruning Strategies 2 and 3) in addition to *tubpn* model (by Pruning Strategy 1). In MHAUIPNU algorithm, the total processing order (<) is defined as follows: (1) items, which are 1-HAUUBIs based on the *tubpn* model, are sorted in *tubpn* ascending order, (2) negative items cannot come before the positive items.

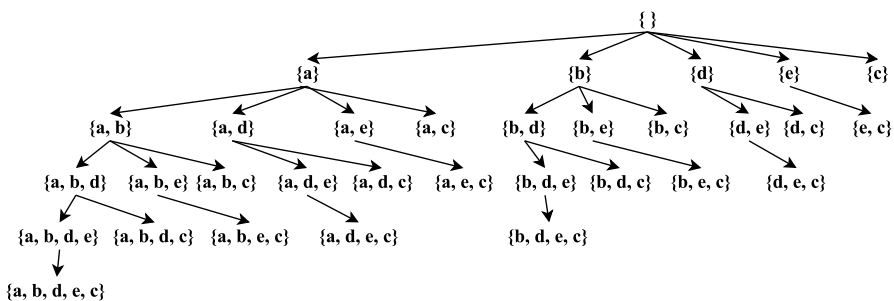


Fig. 1 The enumeration tree of 1-HAUUBIs = {a, b, c, d, e}, where $a < b < d < e < c$

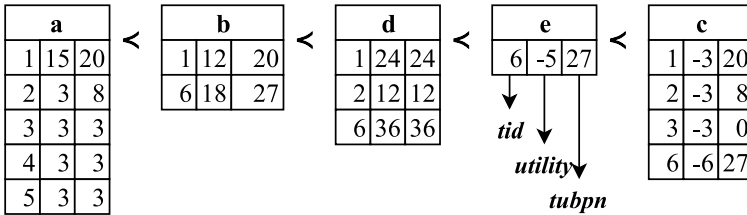


Fig. 2 TUBPNULs of 1-HAUUBIs

For example, based on the *tubpn* model, items *f* and *g* are not 1-HAUUBIs since $tubpn(\{f\}) (= 0.5)$ and $tubpn(\{g\}) (= 8.5)$ are lower than the $minUtil (= 15)$. Therefore, the total processing order is obtained as $a < b < d < e < c$ since $tubpn(\{a\}) = 36 < tubpn(\{b\}) = 46.5 < tubpn(\{d\}) = 72 < tubpn(\{e\}) = 27 < tubpn(\{c\}) = 54$ and items *e* and *c* are negative items.

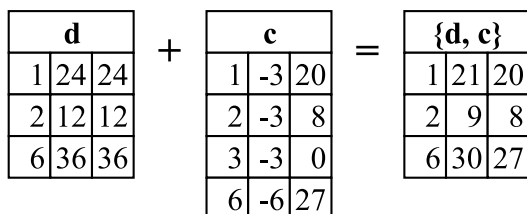
Figure 1 gives the enumeration tree of 1-HAUUBIs based on the total processing order $<$ and Fig. 2 shows the TUBPNU lists of 1-HAUUBIs.

Property 6 (Joining two TUBPNULs) Let X be itemset and $\{X \cup y\}$ be an extension of X by appending an item y to X , such that each item $x < y$ and $\forall x \in X$. The $TUBPNUL(\{X \cup y\})$ can be easily obtained by performing a join operation between $TUBPNUL(\{X\})$ and $TUBPNUL(\{y\})$. The $TUBPNUL(\{X \cup y\})$ is constructed by inserting an element $(E_x.tid, (E_x.utility + E_y.utility), \min\{E_x.tubpn, E_y.tubpn\})$ for each pair of elements $E_x \in TUBPNUL(\{X\})$ and $E_y \in TUBPNUL(\{y\})$, where $E_x.tid = E_y.tid$.

For example, let us construct the $TUBPNUL(\{d, c\})$ by joining $TUBPNUL(\{d\})$ and $TUBPNUL(\{c\})$. Since the list of *tids* of $TUBPNUL(\{d\})$ is (1, 2, 6) and the list of *tids* of $TUBPNUL(\{c\})$ is (1, 2, 3, 6), the intersection is (1, 2, 6). Therefore, the constructed $TUBPNUL(\{d, c\})$ includes three elements with *tids* 1, 2, and 6. The *u* and *tubpn* values of these elements are obtained as discussed in Definitions 2 and 11, respectively. The constructed $TUBPNUL(\{d, c\})$ is presented in Fig. 3.

The pseudo-code of joining two TUBPNULs is given in Construct algorithm (Algorithm 1). The Construct algorithm takes the $TUBPNUL(\{X\})$ and $TUBPNUL(\{y\})$, and a PUA as inputs. The X is the itemset that will be extended and y is the appending item that will extend the itemset X . The input PUA will

Fig. 3 The construction of $TUBPNUL(\{d, c\})$



be used to obtain the sum of positive utilities of itemset $\{X \cup y\}$ if it is not empty. Note that $TUBPNUL(\{X\}).Utility$ and $TUBPNUL(\{X\}).Tubpn$ denote $u(X)$ and $tubpn(X)$, respectively, and $\{X \cup y\}.PU$ denotes the sum of positive utilities of $\{X \cup y\}$. The outputs of the Construct algorithm are $TUBPNUL(\{X \cup y\})$ and $\{X \cup y\}.PU$.

Algorithm 1 Construct($TUBPNUL(\{X\}), TUBPNUL(\{y\}), PUA$)

```

1:  $\{X \cup y\}.PU = 0;$ 
2:  $TUBPNUL(\{X \cup y\}) \leftarrow \text{null};$ 
3:  $i, j \leftarrow 0;$ 
4: while  $i < |TUBPNUL(\{X\})|$  AND  $j < |TUBPNUL(\{y\})|$  do
5:    $E_X \leftarrow i_{th}$  element in the  $TUBPNUL(\{x\});$ 
6:    $E_y \leftarrow j_{th}$  element in the  $TUBPNUL(\{y\});$ 
7:   if  $E_X.tid = E_y.tid$  then
8:      $Element \leftarrow (E_X.Tid, (E_X.u + E_y.u), \min\{E_X.tubpn, E_y.tubpn\})$ 
9:      $TUBPNUL(\{X \cup y\}) \leftarrow Element;$ 
10:     $TUBPNUL(\{X \cup y\}).Utility + = Element.utility$ 
11:     $TUBPNUL(\{X \cup y\}).Tubpn + = Element.tubpn$ 
12:     $i \leftarrow i + 1; j \leftarrow j + 1;$ 
13:    if  $PUA \neq \emptyset$  then
14:       $\{X \cup y\}.PU + = PUA[E_X.tid];$ 
15:    end if
16:  else if  $E_X.tid > E_y.tid$  then
17:     $j \leftarrow j + 1$ 
18:  else
19:     $i \leftarrow i + 1$ 
20:  end if
21: end while
22: return  $TUBPNUL(\{X \cup y\}), \{X \cup y\}.PU;$ 

```

The Construct algorithm works as follows. Let $TUBPNUL(\{X \cup y\})$ be the output list and initially empty (Line 1), and $\{X \cup y\}.PU$ be the sum of positive utilities of itemset $\{X \cup y\}$ and initially equal to 0 (Line 2), and i and j be the first indices in $TUBPNUL(\{X\})$ and $TUBPNUL(\{y\})$, respectively (Line 3). Next, a while loop is run until $i = |TUBPNUL(\{X\})|$ or $j = |TUBPNUL(\{y\})|$ (Lines 4–21). In each loop, the algorithm checks the i_{th} element in $TUBPNUL(\{X\})$ (E_X) and the j_{th} element in $TUBPNUL(\{y\})$ (E_y) if they share the same tid value (Line 7). If so, a new *Element* is constructed (Line 8) and inserted into the $TUBPNUL(\{X \cup y\})$ (Line 9). Then $TUBPNUL(\{X \cup y\}).Utility$ and $TUBPNUL(\{X \cup y\}).Tubpn$ are updated (Lines 10–11), and indices i and j are increased by 1 to check remaining elements in the lists (Line 12). Besides, if PUA is not empty (Line 13), then $\{X \cup y\}.PU$ is updated by adding the value stored in the $(E_X.tid)^{th}$ element of PUA (Line 14). If tid of E_X is greater than the tid of E_y (Line 16) then the index j is increased by 1 to continue with the next element in $TUBPNUL(\{y\})$ (Line 17). Otherwise, the index i is increased by 1 for the same purpose. (Line 19). Finally, the algorithm returns the constructed $TUBPNUL(\{X \cup y\})$ and calculated $\{X \cup y\}.PU$ (Line 22).

Proposed MHAUIPNU Algorithm

In this section, the description of MHAUIPNU algorithm is given first (“[Description of MHAUIPNU Algorithm](#)”). Then an execution trace of MHAUIPNU is presented (“[Execution Trace of the MHAUIPNU Algorithm](#)”). Next, the correctness and completeness of MHAUIPNU algorithm are analyzed (“[Correctness and Completeness of the MHAUIPNU Algorithm](#)”). Finally, the time and space complexity of MHAUIPNU algorithm is discussed (“[Time and Space Complexity Analysis of the MHAUIPNU Algorithm](#)”).

Description of MHAUIPNU Algorithm

The pseudo-code of MHAUIPNU algorithm is given in Algorithm 2. It takes a transactional database DB with internal utilities, an external utility table $EUT(DB)$, and a user-defined minimum utility threshold $minUtil$ as inputs.

Algorithm 2 MHAUIPNU ($DB, EUT(DB), minUtil$)

```

1: scan  $DB$  to determine 1-HAUUBIs based on  $tubpn$  values of items;
2: scan  $DB$  to initialize  $TUBPNUL(\{i\})$  of each item  $i \in$  1-HAUUBIs by ignoring each item  $j \notin$  1-HAUUBIs;
3:  $RTUBPNULs \leftarrow$  each  $TUBPNUL(\{i\})$ , such that  $TUBPNUL(\{i\}).tubpn \geq minUtil$ ;
4: sort  $RTUBPNULs$  based on  $<$  as mentioned by Definition 15;
5:  $MINI = k$ , where  $k$  is the number of  $TUBPNUL$  related to positive items in the  $RTUBPNULs$ .
6: for  $i = 0$  to  $MINI$  do
7:   SearchPEs( $RTUBPNULs, RTUBPNULs[i], i, MINI, minUtil$ )
8: end for

```

The MHAUIPNU algorithm scans the database once to determine the set of 1-HAUUBIs based on $tubpn$ values of items (Line 1). Then it initializes $TUBPNUL$ of each 1-HAUUBI by another database scan (Line 2). Note that, in the second scan, $tubpn$ values of 1-HAUUBIs are re-calculated by ignoring the items which are not 1-HAUUBIs based on the first scan. Thus, $tubpn$ values of some items may be decreased. Therefore, a list named $RTUBPNULs$ (remaining $TUBPNULs$) stores each $TUBPNULs$ related to items if their re-calculated $tubpn$ values satisfy the $minUtil$ (Line 3). $TUBPNULs$ in the $RTUBPNULs$ are sorted based on the processing order $<$ as mentioned in Definition 15 (Line 4). So far, Pruning Strategy 1 is applied.

Then the minimum index of negative items ($MINI$) is assigned as k (Line 5), where k equals to number of $TUBPNUL$ related to positive items in the $RTUBPNULs$. Finally, SearchPEs (Search Positive Extensions) algorithm (Algorithm 3) is called (Lines 6–8) for each item $p \in$ PIs, recursively. Therefore, any itemset consisting any set of items from NIs is directly pruned (Pruning Strategy 2).

SearchPEs Algorithm (Algorithm 3) is designed for the aims of (1) determining HAUIs with only positive items, (2) deciding which HAUIs will be extended with negative items, and (3) deciding itemsets whose positive extensions need to be examined.

Algorithm 3 SearchPEs ($RTUBPNULs$, $TUBPNUL(\{X\})$, i , $MINI$, $minUtil$)

```

1: if  $TUBPNUL(\{X\}).Utility / |X| \geq minUtil$  then
2:   output  $X$  as a HAU;
3:   if  $TUBPNUL(\{X\}).Utility / (|X|+1) > minUtil$  then
4:     generate  $PUA(X)$ ;
5:     SearchNEs( $RTUBPNULs$ ,  $TUBPNUL(\{X\})$ ,  $PUA(X)$ ,  $MINI$ ,  $minUtil$ );
6:     discard  $PUA(X)$ ;
7:   end if
8: end if
9: for  $j = i + 1$  to  $MINI$  do
10:   $TUBPNUL(\{p\}) \leftarrow RTUBPNUL[j]$ ;
11:   $TUBPNUL(\{X \cup p\}) \leftarrow Construct(TUBPNUL(\{X\}), TUBPNUL(\{p\}), \emptyset)$ ;
12:  if  $TUBPNUL(\{X \cup p\}).Tubpn \geq minUtil$  then
13:    SearchPEs ( $RTUBPNULs$ ,  $TUBPNUL(\{X \cup p\})$ ,  $j$ ,  $MINI$ ,  $minUtil$ )
14:  end if
15: end for

```

The SearchPEs algorithm starts with checking whether the $au(X)$ of an itemset X having only positive utility is not lower than the $minUtil$ (Line 1). If so, the itemset X is stored as a HAU (Line 2) and controlled for determining whether negative extensions of X are promising utilizing the Pruning Strategy 3 (Line 3). If negative extensions of X are promising (i.e. if $TUBPNUL(\{X\}).Utility / (|X| + 1) > minUtil$), then the positive utility array of X ($PUA(X)$) is generated (Line 4), and SearchNEs algorithm (Algorithm 4) is called to examine extensions of X with negative items (Line 5). Here, $TUBPNUL(\{X\}).Utility$ equals to $TUBPNUL(\{X\}).PU$ since X is a set of positive items. Once the negative extensions of X are examined, the algorithm discards $PUA(X)$ (Line 6). After that, the algorithm continues to operate remaining lines to examine positive extensions of X . For each positive item p , such that $\forall x \in \{X\} < p$ and $p \in Pls$ (Lines 9–15), the Construct algorithm is called to obtain $TUBPNUL(\{X \cup p\})$ (Line 11). Note that, SearchPEs algorithm calls the Construct algorithm with an empty PUA . The reason is that item p that will extend the itemset X is a positive item, and so $\{X \cup p\}.PU = TUBPNUL(\{X \cup p\}).Utility$. If the $TUBPNUL(\{X \cup p\}).Tubpn$ is not lower than the $minUtil$ (Line 12), the algorithm calls itself (Line 13) to determine if $\{X \cup p\}$ is a HAU and to examine its extensions (Pruning Strategy 1). Otherwise, the itemset $\{X \cup p\}$ will be pruned.

SearchNEs algorithm (Algorithm 4) is designed for the aims of (1) determining HAUs having both positive and negative items and (2) deciding itemsets whose negative extensions need to be examined by utilizing Pruning Strategy 3.

The SearchNEs algorithm examines negative extension of an itemset X via depth-first search utilizing Pruning Strategy 3. For each negative item n , where $\forall x \in X < n$ (Line 1), rest of the lines are executed. The Construct algorithm is called to obtain the $TUBPNUL(\{X \cup n\})$ and $\{X \cup n\}.PU$ (Line 3). Then $au(\{X \cup n\})$ is checked against the $minUtil$ to determine if itemset $\{X \cup n\}$ is a HAU (Line 4). If it is not lower than the $minUtil$, itemset $\{X \cup n\}$ is stored as a HAU (Line 5). Then itemset $\{X \cup n\}$ is controlled for determining whether negative extensions of it are promising by utilizing Pruning Strategy 3. If $minUtil$ is lower than $\{X \cup n\}.PU / (|X| + 1)$ (Line 7), SearchNEs algorithm calls itself (Line 8) to examine the negative extensions of

itemset $\{X \cup n\}$. Otherwise, the negative extensions of itemset $\{X \cup n\}$ will not be examined based on Pruning Strategy 3.

Algorithm 4 SearchNEs ($RTUBPNULs$, $TUBPNUL(\{X\})$, PUA , i , $minUtil$)

```

1: for  $j = i$  to  $|RTUBPNULs|$  do
2:    $TUBPNUL(\{n\}) \leftarrow RTUBPNULs[j]$ ;
3:    $TUBPNUL(\{X \cup n\})$ ,  $\{X \cup n\}.PU \leftarrow Construct(TUBPNUL(\{X\})$ ,
       $TUBPNUL(\{n\})$ ,  $PUA$ );
4:   if  $TUBPNUL(\{X \cup n\}).Utility / |\{X \cup n\}| \geq minUtil$  then
5:     output  $(X \cup n)$  as a HAU;
6:   end if
7:   if  $\{X \cup n\}.PU / (|\{X \cup n\}|+1) > minUtil$  then
8:     SearchNEs( $RTUBPNULs$ ,  $TUBPNUL(\{X \cup n\})$ ,  $PUA$ ,  $j + 1$ ,  $minUtil$ );
9:   end if
10: end for

```

Execution Trace of the MHAUIPNU Algorithm

In this section, the execution trace of MHAUIPNU algorithm is illustrated for the sample database given in Tables 1 and 2. In the execution trace, the $minUtil$ threshold was taken as 15.

The algorithm starts with determining the set of 1-HAUUBIs with the first database scan. As mentioned above, items f and g have lower $tubpn$ values than $minUtil$ ($= 15$) and so the set of 1-HAUUBIs is obtained as $\{a, b, c, d, e\}$. After the second scan, the $TUBPNULs$ of 1-HAUUBIs and the total processing order $<$ are obtained as given in Fig. 2.

Based on total processing order $<$, $MINI$ (the minimum index of negative items) is equal to 3 since there are three positive 1-HAUUBIs which are a , b , and d . The algorithm examines the extensions of positive items a , b , and d by performing a depth-first search strategy. Figure 4 shows the nodes (itemsets) visited by the MHAUIPNU algorithm in the enumeration tree of the running example.

SearchPEs algorithm is first called for the item a . The $au(a)$ is obtained as $TUBPNUL(\{a\}).Utility/|a| = 27$. Since $27 \geq 15$, item a is a HAU. Since item a is a HAU, its negative extensions may include HAU. However, $TUBPNUL(\{a\}).Utility/(|a|+1) = (27/2) = 13.5 \not\geq 15$. Thus, none of negative extensions of item a can be a HAU. The mining process will continue for the positive extensions of item a . The positive items that can extend item a based on the $<$ are items b and d . Since $TUBPNUL(\{a, b\}).Tubpn$ equals to $20 \geq 15$, the itemset $\{a, b\}$ and its any extensions may be HAUIs. Since $au(\{a, b\}) = TUBPNUL(\{a, b\}).Utility/2 = 27/2 \not\geq 15$, the itemset $\{a, b\}$ is not a HAU. Besides, $TUBPNUL(\{a, b\}).Utility/(|\{a, b\}| + 1) = 27/3 \not\geq 15$. Therefore, none of the negative extensions of itemset $\{a, b\}$ can be a HAU. The mining process is continued by extending the itemset $\{a, b\}$ with the positive item d . Since $TUBPNUL(\{a, b, d\}).Tubpn = 20$, the SearchPEs algorithm calls itself for the itemset $\{a, b, d\}$. Since $au(\{a, b, d\})$ is obtained as 17, the itemset $\{a, b, d\}$ is a HAU. However, none of the negative extensions of itemset $\{a, b, d\}$ can be a

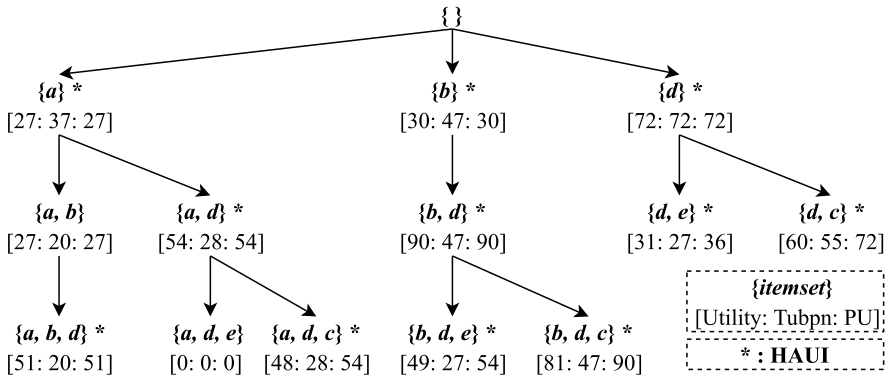


Fig. 4 Visited nodes by the MHAUIPNU in the search space of the running example

HAUI since $TUBPNUL(\{a, b, d\}).Utility / (|\{a, b, d\}| + 1) = 51/4 = 12.75 \not\geq 15$. At this stage, examination of the search space for itemset $\{a, b\}$ is completed.

The next positive item that can extend item a is item d . $TUBPNUL(\{a, d\}).Utility$ and $TUBPNUL(\{a, d\}).Tubpn$ are obtained as 54 and 28, respectively. Therefore, SearchPEs algorithm calls itself for the itemset $\{a, d\}$. Itemset $\{a, d\}$ is a HAUI since $u(\{a, d\}) = 54/2 = 27 \geq 15$. In addition, $TUBPNUL(\{a, d\}).Utility/3 = 54/3 = 18 > 15$, and so the negative extensions of itemset $\{a, d\}$ will be examined. First, itemset $\{a, d, e\}$ will be investigated. Since $TUBPNUL(\{a, d\})$ and $TUBPNUL(\{e\})$ have no elements sharing the same tid value, itemset $\{a, d, e\}$ cannot be a HAUI and so cannot be extended. Then itemset $\{a, d, c\}$ will be investigated. Since $TUBPNUL(\{a, d, c\}).Utility/3 = 48/3 = 16$, itemset $\{a, d, c\}$ is a HAUI. However, there is no remaining negative item for itemset $\{a, d\}$ and item d is the last positive item that can extend item a . Thus, examination of the search space for item a is completed.

Next, same process is performed for item b . Item b is a HAUI since $au(b) = 30$. For item b , item d is the only positive item that can be used to extend it. Therefore, $TUBPNUL(\{b, d\})$ is constructed and $au(\{b, d\})$ is obtained as $= 90/2 = 45 \geq 15$. Itemset $\{b, d\}$ is a HAUI. Since $TUBPNUL(\{b, d\}).Utility/3 = 90/3 = 30 > 15$, the SearchNEs algorithm is called for examining the negative extensions of itemset $\{b, d\}$. For this reason, first, itemset $\{b, d, e\}$ will be investigated. The au of the itemset $\{b, d, e\}$ is obtained as $49/3 \geq 15$. Thus, itemset $\{b, d, e\}$ is a HAUI. Since $\{b, d, e\}.PU/4 = 54/4 \not\geq 15$, examination of the search space for itemset $\{b, d, e\}$ is completed. The next negative item is c that is used to extend the itemset $\{b, d\}$ by SearchNEs algorithm. The $au(\{b, d, c\})$ is 27 and so itemset $\{b, d, c\}$ is a HAUI. Since, c is the last negative item, searching the negative extensions of itemset $\{b, d\}$ is completed. Moreover, item d is the last positive item that can extend item b . Thus, examination of extensions of item b is also completed.

The last positive item is d and $TUBPNUL(\{d\}).Utility$ is 72. Hence, item d is a HAUI and its negative extensions will be investigated. Therefore, after the construction of $TUBPNUL(\{d, e\})$, it is known that $u(\{d, e\})$ is 31. Thus, itemset $\{d, e\}$ is a HAUI since $au(\{d, e\}) = 15.5 = 31/2$. Since $\{d, e\}.PU/3 = 36/3 = 12 \not\geq 15$, itemset

$\{d, e, c\}$ will not be examined. The next negative item is item c . Itemset $\{d, c\}$ is a HAUI since its au value is obtained as $TUBPNUL(\{d, e\}).Utility/2 = 30$. At this stage, there is no remaining negative item to extend item d and item d is the last positive item according to $<$, and so the mining process of the MHAUIPNU algorithm is completed.

Correctness and Completeness of the MHAUIPNU Algorithm

The proposed MHAUIPNU algorithm is designed to discover the correct and complete set of HAUIs in the enumeration tree generated based on the processing order $<$. It searches (or examines) itemsets in the enumeration tree based on the depth-first search strategy. To understand whether an itemset is a HAUI and/or can be pruned, it utilizes the proposed *TUBPNUL* data structure and three pruning strategies. The correctness and completeness of the proposed MHAUIPNU algorithm can be proved by showing that the processing order, data structure, and pruning strategies it uses preserve the correctness and completeness. \square

Lemma 2 *The proposed total processing order preserves the correctness and completeness of the MHAUIPNU algorithm.*

Rationale The proposed processing order is related to how the items are sorted in the enumeration tree. In other words, the processing order determines the parent–child relationship of nodes (itemsets) in the enumeration tree. In this study, the enumeration tree constructed based on the total processing order $<$ on remaining items after the item whose *tubpn* values are lower than the *minUtil* is pruned by Pruning Strategy 1. Pruning Strategy 1 ensures the correctness and completeness of the MHAUIPNU algorithm as mentioned by Lemma 3. Therefore, it is clear that the enumeration tree of the search space generated by the total processing order includes all possible HAUIs. As a result, the proposed total processing order $<$ preserves the correctness and completeness of the MHAUIPNU algorithm. \square

Lemma 3 *The proposed TUBPNUL data structure preserves the correctness and completeness of the MHAUIPNU algorithm.*

Rationale *The proposed TUBPNUL data structure is designed to store utility and tubpn values of itemsets. When searching (or examining) the itemsets in the enumeration tree in depth-first search manner, their TUBPNULs can be easily and correctly constructed as mentioned by Property 6. TUBPNUL data structure is efficient to obtain values that are used by Pruning Strategy 1 and Pruning Strategy 3. Besides, average-utility of an itemset X can be easily derived from its $TUBPNUL(\{X\})$, i.e., $au(X) = TUBPNUL(\{X\}).Utility / |X|$, and so any examined itemsets can be easily determined as a HAUI or not by the MHAUIPNU algorithm. As a result, the proposed TUBPNUL data structure preserves the completeness and correctness of the MHAUIPNU algorithm.* \square

Lemma 4 *Pruning Strategy 1 preserves the correctness and completeness of the MHAUIPNU algorithm.*

Rationale We know that the average-utility of any itemset X cannot be greater than its $tubpn$ value (Theorem 1). We also know that, for any pair of itemsets X and Y , such that $X \subseteq Y$, $tubpn(X) \geq tubpn(Y)$ holds (Property 2). As a result, pruning the search space by utilizing Pruning Strategy 1 preserves the completeness and correctness of the MHAUIPNU algorithm since it is clear that if $tubpn(X) \leq minUtil$ holds, then X not a HAU and none of its extension can be a HAU. \square

Lemma 5 *Pruning Strategy 2 preserves the correctness and completeness of the MHAUIPNU algorithm.*

Rationale Since a HAU should contain at least one positive item (Lemma 1), the itemsets which have only negative items cannot be a HAU. Thanks to proposed processing order, Pruning Strategy 2 eliminates all these itemsets directly from the enumeration tree. As a result, Pruning Strategy 2 preserves the completeness and correctness of the MHAUIPNU algorithm. \square

Lemma 6 *Pruning Strategy 3 preserves the correctness and completeness of the MHAUIPNU algorithm.*

Rationale We know that $(X.PU/(|X| + 1)) > au(XN)$ holds for an itemset X and any of its negative extensions XN (Property 4). Thus, it is true that none of negative extensions of X can be a HAU if $(X.PU/(|X| + 1)) > minUtil$ holds. Based on the proposed processing order, Pruning Strategy 3 eliminates all the negative extensions of X directly from the enumeration tree if $(X.PU/(|X| + 1)) > minUtil$ holds. As a result, Pruning Strategy 3 preserves the correctness and completeness of the MHAUIPNU algorithm. \square

Theorem 2 *The MHAUIPNU algorithm is correct and complete. Based on a given $minUtil$ value, it discovers the correct and complete set of HAUIs in a given dataset containing items with negative utilities.*

Proof The proposed MHAUIPNU algorithm is designed to discover the HAUIs utilizing the enumeration tree generated based on the total processing order \prec , together with three pruning strategies (Pruning Strategies 1, 2, and 3). For each itemset that MHAUIPNU visits in the enumeration tree (or for each itemsets which is not pruned by the pruning strategies), MHAUIPNU constructs its $TUBPNUL$ data structure to determine it is a HAU or not. Therefore, based on Lemmas 2, 3, 4, 5, and 6, it can be said that the set of discovered itemsets by MHAUIPNU algorithm in a dataset containing items with negative utilities is complete and correct. \square

Time and Space Complexity Analysis of the MHAUIPNU Algorithm

The time and space complexity of an algorithm can be defined as the time and space required in the worst case, respectively. In this section, we analyze the time and space complexity of the MHAUIPNU algorithm for the worst case scenario.

Let T , I , P , and N be number of transactions, total number of items, number of positive items, and number of negative items in a given database, respectively. The MHAUIPNU algorithm scans the database once to obtain *tubpn* values of items. Thus, the time complexity in the first database scan is $\mathcal{O}(T \times I)$. In the second database scan, initial *TUBPNULs* are generated for promising items. In the worst case, none of items has *tubpn* value lower than the *minUtil*, which means that all the items are promising. Therefore, the runtime complexity of the second database is also $\mathcal{O}(T \times I)$. Then before embarking to start searching HAUIs, *TUBPNULs* are sorted based on the total processing order \prec . Based on the total processing order \prec (Definition 15), we know that items are sorted in *tubpn* ascending order and a negative item always come after from all the positive items. Therefore, it is sufficient to sort the *TUBPNULs* of the positive items and the *TUBPNULs* negative items among themselves. For sorting, any method can be used. Assume that quick sort method is applied. The time complexity is $\mathcal{O}(P \log 2P)$ and $\mathcal{O}(N \log 2N)$ for sorting *TUBPNULs* of the positive items and the *TUBPNULs* negative items, respectively. Therefore, for the worst case scenario, the MHAUIPNU algorithm requires $\mathcal{O}((2 \times T \times I + P \log 2P + N \log 2N)$ time before embarking to search HAUIs (or examine the enumeration tree of the search space). Besides, the worst case scenario assumes that each item exists in each transaction. Therefore, the MHAUIPNU algorithm requires $\mathcal{O}(I \times T \times 3)$ space for storing initial *TUBPNULs* in the memory.

For searching HAUIs, the MHAUIPNU algorithm constructs *TUBPNULs* of itemsets while traversing the enumeration tree generated based on the processing order \prec . To construct *TUBPNUL* of an itemset $\{X, a, b\}$, it is needed to compare *TUBPNULs*' entries of two related itemset $\{X, a\}$ and itemset $\{X, b\}$ to find the entries registered for the same transaction. In the worst case, *TUBPNUL* of an itemset contains T entries based on the assumption that each item exists in each transaction. Since comparing two *TUBPNULs* is done in linear time, the time complexity of constructing *TUBPNUL* of an itemset is $\mathcal{O}(T)$. Besides, to prune itemset by Pruning Strategy 3, the algorithm use a one-dimensional array, called the positive utility array (*PUA*), of size T . In the MHAUIPNU algorithm, *PUA* of an itemset will be generated if it has only positive items and its negative extensions are promising. Besides, *PUA* of an itemset is discarded right after its all negative extensions are examined. Thus, storing *PUA* requires $\mathcal{O}(T)$ space. Since a *PUA* is generated in linear time, the time complexity of generating a *PUA* is $\mathcal{O}(T)$ time for the worst case.

We know that there are $2^I - 1$ itemsets in the search space. However, the MHAUIPNU algorithm does not construct *TUBPNULs* of $2^N - 1$ k -itemsets, where $k \geq 2$, which consists of only negative items, by utilizing the Pruning Strategy 2. As a result, Pruning Strategy 2 reduces the number of itemsets in the enumeration tree to $2^I - 2^N - 1$. For the worst case, it is assumed that none of them can be pruned by either Pruning Strategy 1 or Pruning Strategy 3. Therefore, $2^I - 2^N - 1$ *TUBPNULs* should be constructed, and $2^P - 1$ *PUA* should be generated while searching

Table 3 Compared algorithms

Algorithm	Upper bound	Processing order	Pruning strategies
Naïve-auub ^{pn}	<i>auub^{pn}</i>	<i>auub^{pn}</i> -ascending	based on <i>auub^{pn}</i> model
Naïve-tubpn	<i>tubpn</i>	<i>tubpn</i> -ascending	Pruning Strategy 1
MHAUIPNU	<i>tubpn</i>	< (Definition 15)	Pruning Strategies 1, 2, and 3

Table 4 Dataset parameters

Parameter	Explanation
# <i>Ts</i>	The total number of transaction
# <i>PIs</i>	The total number of unique positive items
# <i>NI</i> s	The total number of unique negative items
<i>avgL</i>	The average length of transactions
<i>Density</i>	The density of the dataset $\left\langle \frac{avgL}{\#PIs + \#NI} \times 100 \right\rangle$

HAUIs. Therefore, for the MHAUIPNU algorithm, the worst case time complexity is $\mathcal{O}(2 \times T \times I) + \mathcal{O}(P \log 2P + N \log 2N) + \mathcal{O}(T \times (2^I - 2^N - 2)) + \mathcal{O}(T \times (2^P - 1))$ and the worst case memory usage is $\mathcal{O}((2^I - 2^N - 2) \times T \times 3) + \mathcal{O}(T)$.

On the other hand, the worst case scenario does not usually occur. In general, the time and space complexity of the MHAUIPNU algorithm can be considered as pseudo-polynomial since the complexity of time and space for each itemset is close to linear and the number of visited itemsets in the search space is conditional on the effectiveness of the pruning strategies for the given dataset.

Experimental Result

In this section, the performance evaluation of the MHAUIPNU algorithm is given. To evaluate effectiveness of proposed *tubpn* model by comparing with *auub^{pn}* model, two algorithms are also designed, which are named as Naïve-auub^{pn} and Naïve-tubpn algorithms based on *auub^{pn}* and *tubpn* models, respectively (Table 3). They use their downward-closure properties to prune the search space. The Naïve-tubpn algorithm is also used to compare with the MHAUIPNU algorithm to evaluate the effect of pruning strategies (Pruning Strategies 2 and 3) related to items with negative utilities (Table 3).

We compare the algorithms in terms of runtime, the number of total visited nodes, and memory usage. We also evaluate the effect of the number of negative items in databases on the performance of algorithms. Performance analysis of the algorithms in terms of runtime, the total number of visited nodes, and memory usage are evaluated using six real datasets which are obtained from the open source data mining library, SPMF [4]. The statical information of the real datasets are provided in Table 5. Table 4 gives the explanations of the parameters used in Table 5.

Table 5 Real datasets

Dataset	#Ts	#PIs	#NIs	AvgL	Density
<i>Chess</i>	3196	37	38	37	49.3333
<i>Mushroom</i>	8124	74	45	23	19.3277
<i>Accidents</i>	340,183	230	238	33.8	7.2222
<i>Pumsb</i>	49,046	1079	1034	74	3.5021
<i>Retail</i>	88,162	8223	8247	10.3	0.0625
<i>Kosarak</i>	990,002	20,700	20,570	8.1	0.0196

Besides, two synthetic datasets are used for evaluation of the number of negative items on the performances of the algorithms.

The algorithms are implemented using Java programming language. All the experiments are performed on a computer equipped with an i5-5200U 2.2 GHz processor and 8 GBs of RAM.

Runtime

In this experiment, the runtime performances of the algorithms are compared. All the algorithms were run on each dataset given in Table 5 under various *minUtil* thresholds. Figure 5 presents the runtime results in log scale, except for the Retail dataset.

As can be seen in Fig. 5, the algorithms need more time to perform their mining tasks for each dataset as the *minUtil* value decreases. This is because as the *minUtil* gets lower, the datasets will have more promising itemsets, and thus the search space will expand. Figure 5 also indicates that Naïve-tubpn are always faster than the Naïve-auub^{pn}. This is reasonable since *tubpn* values of itemsets are tighter than their *auub^{pn}* values. Thus, Naïve-tubpn prunes the search space more effectively than Naïve-auub^{pn}. Moreover, when the runtimes of Naïve-tubpn and MHAUIPNU are compared, it can be seen that Pruning Strategies 2 and 3 of MHAUIPNU algorithm have a significant role on enhancing the performance of MHAUIPNU. For example, when the *minUtil* is set to 60×10^3 for the Chess (Fig. 5a), MHAUIPNU, Naïve-tubpn, and Naïve-auub^{pn} terminate their mining tasks in 0.5 seconds, 157 seconds, and 351 seconds, respectively. Note that, we could not present the runtimes of Naïve-auub^{pn} and Naïve-tubpn when *minUtil* is set to 16×10^5 (or lower values) and 14×10^5 (or lower values), respectively, for the Pumsb dataset. The reason is that they require long execution times (more than 10,000 s) for the above-mentioned settings. On the other hand, MHAUIPNU needs only 118 seconds to complete the mining task for the Pumsb dataset when the *minUtil* is set to 14×10^5 . As can be seen, MHAUIPNU significantly outperforms Naïve-auub^{pn} and Naïve-tubpn thanks to *tubpn* model (Pruning Strategy 1) and properties related to items with negative utilities (Pruning Strategies 2 and 3). It is also observed from the experiments that MHAUIPNU is up to three orders of magnitude faster than Naïve-tubpn and up to four orders of magnitude faster than Naïve-auub^{pn}. As a result, the proposed

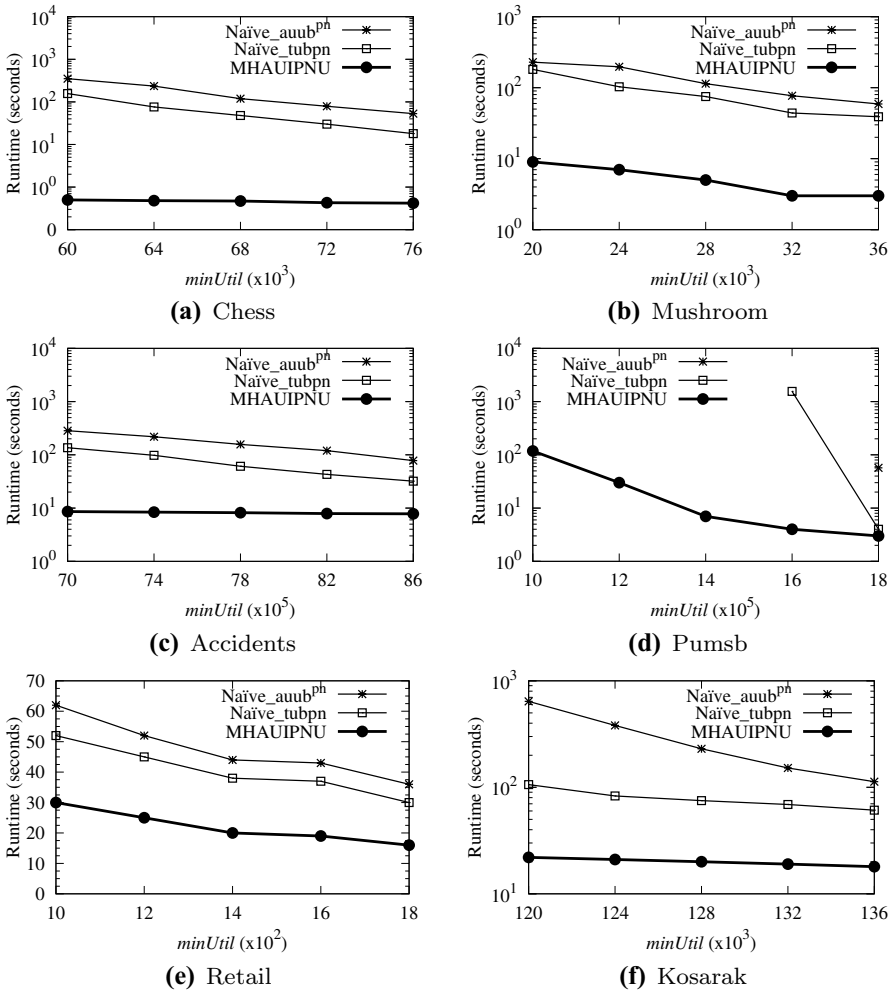


Fig. 5 Runtimes for various *minUtils*

MHAUIPNU is computationally efficient on solving the problem of HAUIM with positive and negative utilities.

The Number of 1-HAUUBIs and Visited Nodes

In this experiment, the number of 1-HAUUBIs obtained by *tubpn* and *aub^{pn}* models are first compared. Then the number of nodes (itemsets) visited by the algorithms are compared for to understand their runtime performances.

Results related to obtained 1-HAUUBIs by *tubpn* and *aub^{pn}* models are given in Fig. 6. As can be seen in Fig. 6, *tubpn* produces less 1-HAUUBIs than the *aub^{pn}* model. This is because *tubpn* values of itemsets are tighter than their *aub^{pn}*

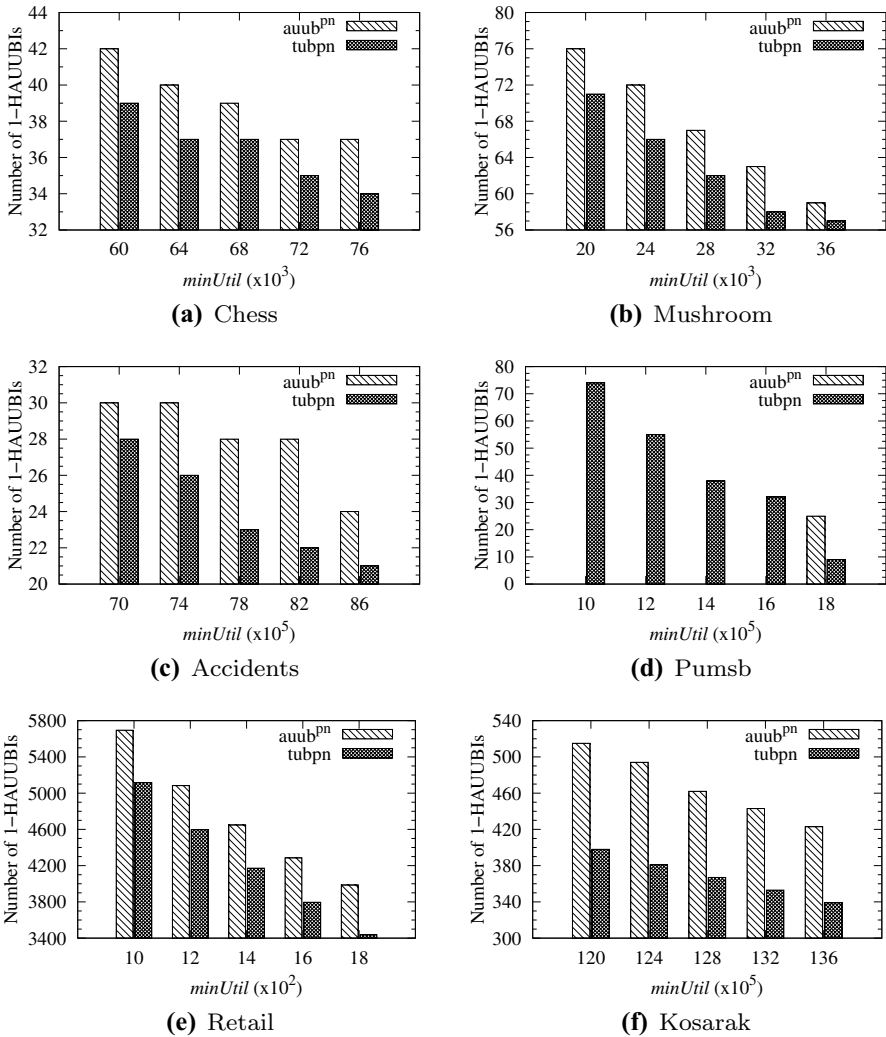


Fig. 6 The number of 1-HAUUBIs obtained by *tubpn* and *aub^{pn}* for various *minUtils*

values as discussed with Property 3. Particularly, if datasets have large number of items, such as Kosarak (Fig. 6f) and Retail (Fig. 6e), the performance of *tubpn* to reduce the number of 1-HAUUBIs compared to *aub^{pn}* is very considerable.

Results related to the number of nodes visited by the algorithms are given in Fig. 7. A visited node (itemset) means the node which is examined in the enumeration tree of the search space. As can be seen in Fig. 7, Naïve-*tubpn* visits one to three magnitude less nodes compared to Naïve-*aub^{pn}*. The reasons as follows: (1) *tubpn* produces less 1-HAUUBIs than the *aub^{pn}* model and (2) *tubpn* values of itemsets are always lower than their *aub^{pn}* values. Thus, it can be said that *tubpn* model is more efficient than the *aub^{pn}* model on reducing the search space.

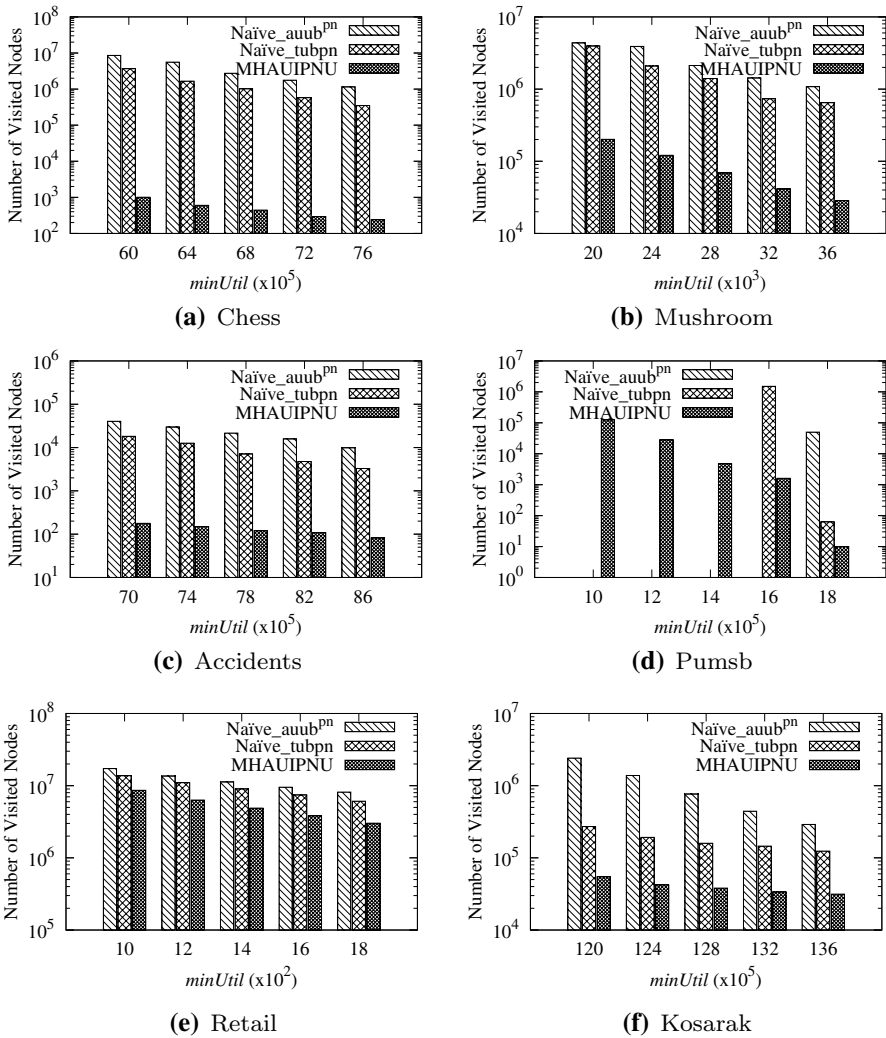


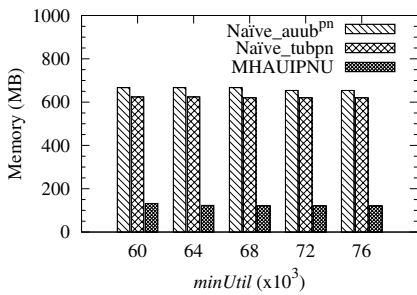
Fig. 7 Number of visited nodes for various *minUtils*

Figure 7 indicates that the proposed MHAUIPNU dramatically reduces the number of visited nodes thanks to its Pruning Strategy 2 and Pruning Strategy 3 in addition to usage of *tubpn* model (Pruning Strategy 1). The proposed *tubpn* is efficient to decrease the upper bounds of itemsets. Besides, itemsets containing only negative items are directly pruned by Pruning Strategy 2 and the positive utilities of itemsets are directly used to prune the their negative extensions by Pruning Strategy 3. For example, on Chess dataset (Fig. 7a), when the *minUtil* is set to 60×10^3 , the number of visited nodes by Naive-aub^{pn} and Naive-tubpn is, respectively, 8478 and 3655 times greater than the number of nodes visited by MHAUIPNU.

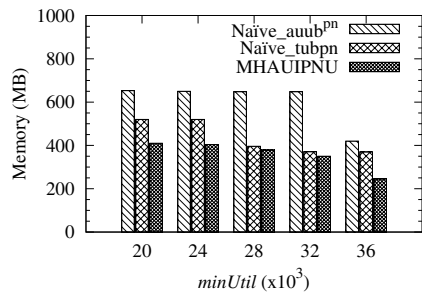
Memory Usage

In this experiment, the memory usage of the algorithms are compared. The results are shown in Fig. 8.

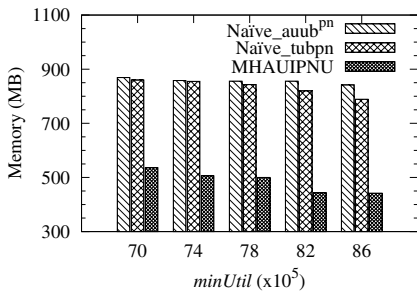
It is observed that memory consumptions of algorithms increase as the $minUtil$ value decreases for each datasets. As the $minUtil$ decreases, more 1-HAUUBIs are generated and more HAUIs are discovered. Hence, more itemset in the enumeration tree of the search space are visited, and so memory needs increase. It is also observed that MHAUIPNU and Naïve-tubpn need less memory than Naïve-aaub^{pn}. This is reasonable because *aub^{pn}* model cannot prune more itemsets than



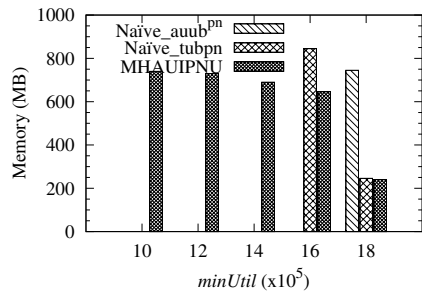
(a) Chess



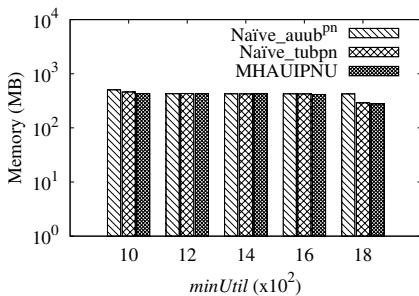
(b) Mushroom



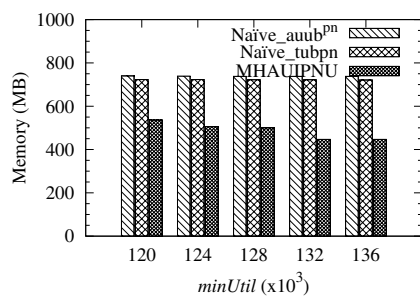
(c) Accidents



(d) Pumsb



(e) Retail



(f) Kosarak

Fig. 8 Memory usages for various $minUtil$ s

the *tubpn* model. As can be seen in Fig. 8, MHAUIPNU consumes the least memory usage.

Effect of the Number of Negative Items

In this experiment, the effects of the number of negative items in databases is evaluated on the performance of the algorithms. For evaluation, two synthetic datasets, called *c20d10k* and *t20i6d100k*, are used. They were obtained from the SPMF [4], and their statical information are given in Table 6.

These datasets contain only binary information since they are generated for frequent itemset mining. In this study, to make them as quantitative transactional databases, the external utilities of items were generated using a Gaussian distribution where the mean is 50 and internal utilities of items in each transaction were randomly generated in the range of [1, 5]. Then we generated five different datasets from these synthetic datasets, each containing 30%, 40%, 50%, 60%, and 70% negative items. To make a database contain 30% negative items, 30% items among all unique items was randomly selected and their external utilities were multiplied by -1 . Then to make a database contain 40% negative items a new set of 10% items was also randomly selected and their external utilities were also multiplied by -1 . To ensure that the datasets contain 50%, 60%, and 70% negative items, the same procedure was followed.

To evaluate the effect of the number of negative items on the performance of algorithms, their runtimes and total number of visited nodes are compared for each dataset by changing the number of negative items from 30% to 70% of all unique items. For each experiments, $minUtil$ is fixed to 10^5 . Experimental results are given in Figs. 9 and 10 for the *c20d10k* and *t20i6d100k* datasets, respectively.

Experimental evaluations show that as the number of negative items increases in the datasets, the algorithms need less time to perform the mining task (Figs. 9a, 10a), and visit less nodes (Figs. 9b, 10b). The experiments also indicate that the runtime gap between Naïve-auub^{pn} and Naïve-tubpn increases, as the number of negative items increases. This shows that Pruning Strategy 1 is becoming more effective with the increase of negative items in the datasets.

Moreover, it is also observed that the performance of the proposed MHAUIPNU algorithm increases with the increase of the number of negative items in the datasets, in terms of runtime and the total number of visited nodes. This shows that when the size of negative items increases in databases, Pruning Strategy 2 and Pruning Strategy 3 prune the search space more effectively.

For example, when the number of negative items is 30% in the *c20d10k* dataset, MHAUIPNU algorithm is about 5 and 4 times faster than the Naïve-auub^{pn} and Naïve-tubpn, respectively, and the number of visited nodes by MHAUIPNU is about 7 and

Table 6 Synthetic datasets

Dataset	#Ts	#items	AvgL	Density
<i>c20d10k</i>	10,000	192	20	10.4166
<i>t20i6d100k</i>	99,922	843	19.89	2.3594

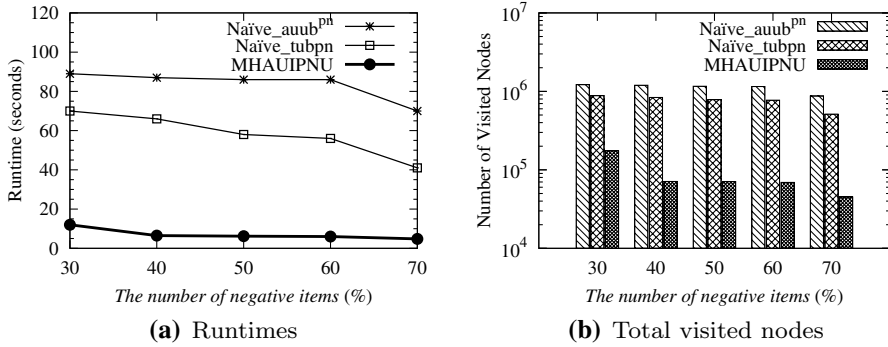


Fig. 9 The effects of the number of negative items in *c20d10k* dataset on the performance of algorithms

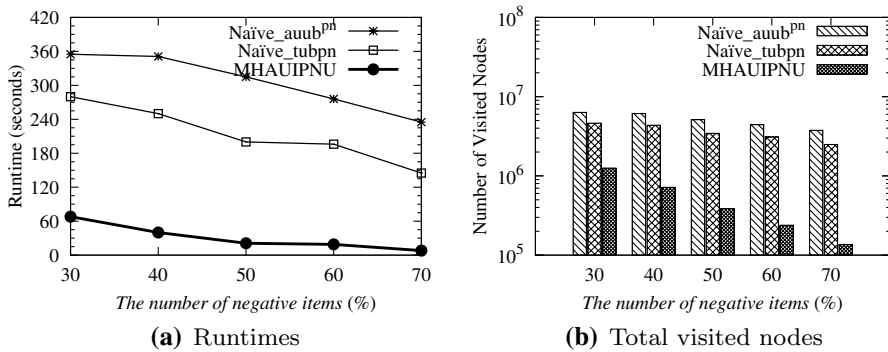


Fig. 10 The effect of the number of negative items in *t20i6d100k* dataset on the performance of algorithms

5 times less than the Naïve-aaub^{pn} and Naïve-tubpn, respectively. However, when the number of negative items is increased to 70% in the *c20d10k* dataset, MHAUIPNU algorithm is about 30 and 18 times faster than the Naïve-aaub^{pn} and Naïve-tubpn, respectively, and the number of visited nodes by MHAUIPNU is about 20 and 11 times less than Naïve-aaub^{pn} and Naïve-tubpn, respectively. Similar results are obtained for the *t20i6d100k* dataset.

Conclusion and Future Works

High-average-utility itemset mining (HAUIM) is important for many real-world application areas. It takes into account internal and external utilities (such as unit quantities and unit profits) of the itemsets.

To the best of our knowledge, in the literature, there is no HAUIM algorithm designed for mining HAUIs out of databases containing both positive and negative utilities. This paper proposes an upper bound named *tubpn* with its pruning strategy

to prune itemsets. Besides, two other pruning strategies also proposed utilizing the properties related to items with negative utilities. To store the required information and efficiently calculate average-utilities and *tubpn* values of itemsets, a *TUBPNUL* data structure is also developed. Then an algorithm called MHAUIPNU is presented to solve the problem of HAUIM with positive and negative items, utilizing three proposed pruning strategies and the *TUBPNUL*.

To evaluate the efficiency of proposed *tubpn* and the pruning strategies, two algorithms are also implemented. Experimental analysis shows that the proposed MHAUIPNU algorithm mines HAUIs out of the database having positive and negative utilities, efficiently. It is also shown that pruning strategies which are related to items with negative utilities are very efficient on pruning the search space of itemsets with negative utilities. The results show that the proposed MHAUIPNU algorithm outperforms other algorithms as the number of negative items increases and/or the minimum utility threshold decreases.

As a future work, we would like to propose more efficient algorithms by introducing more efficient upper bounds and data structures to enhance the efficiency of solving the problem in terms of runtime and memory consumption. Besides, developing efficient incremental and interactive mining HAUIs out of databases having positive and negative utilities is another topic that can be studied in future.

References

1. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. *ACM SIGMOD Rec.* **22**(2), 207–216 (1993). <https://doi.org/10.1145/170036.170072>
2. Chu, C.J., Tseng, V.S., Liang, T.: An efficient algorithm for mining high utility itemsets with negative item values in large databases. *Appl. Math. Comput.* **215**(2), 767–778 (2009). <https://doi.org/10.1016/j.amc.2009.05.066>
3. Deng, Z.H.: DiffNodesets: an efficient structure for fast mining frequent itemsets. *Appl. Soft. Comput.* **41**, 214–223 (2016). <https://doi.org/10.1016/j.asoc.2016.01.010>
4. Fournier-Viger, P., Gomariz, A., Gueniche, T., Soltani, A., Wu, C.W., Tseng, V.S.: Spmf: a java open-source pattern mining library. *J. Mach. Learn. Res.* **15**, 3389–3393 (2014)
5. Fournier-Viger, P., Wu, C.W., Zida, S., Tseng, V.S.: FHM: faster high-utility itemset mining using estimated utility co-occurrence pruning. In: *Lect. Notes in Comput. Sci.*, pp. 83–92. Springer International Publishing (2014). https://doi.org/10.1007/978-3-319-08326-1_9
6. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. *ACM SIGMOD Rec.* **29**(2), 1–12 (2000). <https://doi.org/10.1145/335191.335372>
7. Hong, T.P., Lee, C.H., Wang, S.L.: Effective utility mining with the measure of average utility. *Expert Syst. with Appl.* **38**(7), 8259–8265 (2011). <https://doi.org/10.1016/j.eswa.2011.01.006>
8. Huang, H., Wu, X., Relue, R.: Mining frequent patterns with the pattern tree. *New Gener. Comput.* **23**(4), 315–337 (2005). <https://doi.org/10.1007/bf03037636>
9. Kim, D., Yun, U.: Efficient algorithm for mining high average-utility itemsets in incremental transaction databases. *Appl. Intell.* **47**(1), 114–131 (2017). <https://doi.org/10.1007/s10489-016-0890-z>
10. Krishnamoorthy, S.: Pruning strategies for mining high utility itemsets. *Expert Syst. Appl.* **42**(5), 2371–2381 (2015). <https://doi.org/10.1016/j.eswa.2014.11.001>
11. Krishnamoorthy, S.: Efficiently mining high utility itemsets with negative unit profits. *Knowl. Based Syst.* **145**, 1–14 (2018). <https://doi.org/10.1016/j.knsys.2017.12.035>
12. Lan, G.C., Hong, T.P., Tseng, V.S.: Efficiently mining of high average-utility itemsets with an improved upper-bound strategy. *Int. J. Inf. Technol. Decis. Making* **11**(05), 1009–1030 (2012). <https://doi.org/10.1142/s0219622012500307>
13. Lan, G.C., Hong, T.P., Tseng, V.S.: A projection-based approach for discovering high average-utility itemsets. *J. Inf. Sci. Eng.* **28**, 193–209 (2012)

14. Lin, C.W., Hong, T.P., Lu, W.H.: Efficiently mining high average utility itemsets with a tree structure. In: *Intell. Inf. Database Syst.*, pp. 131–139. Springer, Berlin (2010). https://doi.org/10.1007/978-3-642-12145-6_14
15. Lin, C.W., Hong, T.P., Lu, W.H.: Using the structure of prelarge trees to incrementally mine frequent itemsets. *New Gener. Comput.* **28**(1), 5–20 (2010). <https://doi.org/10.1007/s00354-008-0072-6>
16. Lin, C.W., Hong, T.P., Lu, W.H.: An effective tree structure for mining high utility itemsets. *Expert Syst. Appl.* **38**(6), 7419–7424 (2011). <https://doi.org/10.1016/j.eswa.2010.12.082>
17. Lin, J.C.W., Fournier-Viger, P., Gan, W.: FHN: an efficient algorithm for mining high-utility itemsets with negative unit profits. *Knowl. Based Syst.* **111**, 283–298 (2016). <https://doi.org/10.1016/j.knosys.2016.08.022>
18. Lin, J.C.W., Li, T., Fournier-Viger, P., Hong, T.P., Zhan, J., Voznak, M.: An efficient algorithm to mine high average-utility itemsets. *Adv. Eng. Inf.* **30**(2), 233–243 (2016). <https://doi.org/10.1016/j.aei.2016.04.002>
19. Lin, J.C.W., Ren, S., Fournier-Viger, P., Hong, T.P.: EHAUPM: efficient high average-utility pattern mining with tighter upper bounds. *IEEE Access* **5**, 12927–12940 (2017). <https://doi.org/10.1109/access.2017.2717438>
20. Lin, J.C.W., Ren, S., Fournier-Viger, P., Hong, T.P., Su, J.H., Vo, B.: A fast algorithm for mining high average-utility itemsets. *Appl. Intell.* **47**(2), 331–346 (2017). <https://doi.org/10.1007/s10489-017-0896-1>
21. Lin, J.C.W., Shao, Y., Fournier-Viger, P., Djenouri, Y., Guo, X.: Maintenance algorithm for high average-utility itemsets with transaction deletion. *Appl. Intell.* **48**(10), 3691–3706 (2018). <https://doi.org/10.1007/s10489-018-1180-8>
22. Liu, J., Wang, K., Fung, B.C.: Mining high utility patterns in one phase without generating candidates. *IEEE Trans. Knowl. Data Eng.* **28**(5), 1245–1257 (2016). <https://doi.org/10.1109/tkde.2015.2510012>
23. Liu, M., Qu, J.: Mining high utility itemsets without candidate generation. In: *Proc. of the 21st ACM Int. Conf. Inf. Knowl. Manag., CIKM* (2012). <https://doi.org/10.1145/2396761.2396773>
24. Liu, Y., Liao, W.K., Choudhary, A.: A two-phase algorithm for fast discovery of high utility itemsets. In: *Adv. Knowl. Discov. Data Min.*, pp. 689–695. Springer, Berlin (2005). https://doi.org/10.1007/11430919_79
25. Lu, T., Vo, B., Nguyen, H.T., Hong, T.P.: A new method for mining high average utility itemsets. In: *Comput. Inf. Syst. Ind. Manag.*, pp. 33–42. Springer, Berlin (2014). https://doi.org/10.1007/978-3-662-45237-0_5
26. Peng, A.Y., Koh, Y.S., Riddle, P.: mHUIminer: a fast high utility itemset mining algorithm for sparse datasets. In: *Adv. in Knowl. Discov. Data Min.*, pp. 196–207. Springer International Publishing (2017). https://doi.org/10.1007/978-3-319-57529-2_16
27. Ryang, H., Yun, U.: Indexed list-based high utility pattern mining with utility upper-bound reduction and pattern combination techniques. *Knowl. Inf. Syst.* **51**(2), 627–659 (2016). <https://doi.org/10.1007/s10115-016-0989-x>
28. Singh, K., Shakya, H.K., Singh, A., Biswas, B.: Mining of high-utility itemsets with negative utility. *Expert Syst.* (2018). <https://doi.org/10.1111/exsy.12296>
29. Truong, T., Duong, H., Le, H.B., Viger, P.F.: Efficient vertical mining of high average-utility itemsets based on novel upper-bounds. *IEEE Trans. Knowl. Data Eng.*, pp. 301–314 (2018). <https://doi.org/10.1109/tkde.2018.2833478>
30. Tseng, V.S., Shie, B.E., Wu, C.W., Yu, P.S.: Efficient algorithms for mining high utility itemsets from transactional databases. *IEEE Trans. Knowl. Data Eng.* **25**(8), 1772–1786 (2013). <https://doi.org/10.1109/tkde.2012.59>
31. Tseng, V.S., Wu, C.W., Shie, B.E., Yu, P.S.: UP-growth: an efficient algorithm for high utility itemset mining. In: *Proc. 16th ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.* (2010). <https://doi.org/10.1145/1835804.1835839>
32. Wu, J.M.T., Lin, J.C.W., Pirouz, M., Fournier-Viger, P.: TUB-HAUPM: tighter upper bound for mining high average-utility patterns. *IEEE Access* **6**, 18655–18669 (2018). <https://doi.org/10.1109/access.2018.2820740>
33. Wu, T.Y., Lin, J.C.W., Shao, Y., Fournier-Viger, P., Hong, T.P.: Updating the discovered high average-utility patterns with transaction insertion. In: *Adv. Intell. Syst. Comput.*, pp. 66–73. Springer Singapore (2017). https://doi.org/10.1007/978-981-10-6487-6_9

34. Yildirim, I., Celik, M.: FIMHAUI: Fast incremental mining of high average-utility itemsets. In: 2018 Int. Conf. on Artif. Intell. and Data Process. (IDAP). IEEE (2018). <https://doi.org/10.1109/idap.2018.8620819>
35. Yun, U., Kim, D.: Mining of high average-utility itemsets using novel list structure and pruning strategy. *Future Gener. Comput. Syst.* **68**, 346–360 (2017). <https://doi.org/10.1016/j.future.2016.10.027>
36. Yun, U., Kim, D., Yoon, E., Fujita, H.: Damped window based high average utility pattern mining over data streams. *Knowl. Based Syst.* **144**, 188–205 (2018). <https://doi.org/10.1016/j.knsys.2017.12.029>
37. Zida, S., Fournier-Viger, P., Lin, J.C.W., Wu, C.W., Tseng, V.S.: EFIM: a fast and memory efficient algorithm for high-utility itemset mining. *Knowl. Inf. Syst.* **51**(2), 595–625 (2016). <https://doi.org/10.1007/s10115-016-0986-0>

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

Irfan Yildirim received B.Sc. degree in computer engineering from Kocaeli University, Kocaeli, Turkey, in 2009 and M.Sc. degree in computer science from the City College of New York, New York, NY, USA, in 2014. He is currently pursuing Ph.D. degree in computer engineering at Erciyes University, Kayseri, Turkey. His research interests include pattern mining, big data, and spatio-temporal data mining.

Mete Celik received B.Sc. degree in control and computer engineering and M.Sc. degree in electrical engineering from Erciyes University, Kayseri, Turkey, in 1999 and 2001, respectively, and Ph.D. degree in computer science from the University of Minnesota, Minneapolis, USA, in 2008. He is currently a faculty member of the Department of Computer Engineering, Erciyes University, Turkey. His research interests include data analysis, big data, spatial databases, spatial data mining, spatio-temporal data mining, and location-based services. He is a member of the IEEE and ACM.