Journal of AI and Data Mining
Vol 7, No 4, 2019, 537-550
DOI: 10.22044/JADM.2019.7483.1893

A New Algorithm for High Average-utility Itemset Mining

A. Soltani[1]* and M. Soltani[2]

1. Dept. of Computer Engineering, University of Bojnord, Bojnord, Iran.
2. Dept. of Computer Engineering, Quchan University of Technology, Quchan, Iran.

Received 22 September 2018; Revised 22 January 2019; Accepted 07 April 2019
**Corresponding author: a.soltani@ub.ac.ir (A.Soltani).*

Abstract
High utility itemset mining (HUIM) is a new emerging field in data mining, which has gained growing interests due to its various applications. The goal of this work is to discover all itemsets whose utility exceeds minimum threshold. The basic HUIM problem does not consider length of itemsets in its utility measurement and the utility values tend to become higher for itemsets containing more items. Hence, HUIM algorithms discover a huge enormous number of long patterns. High average-utility itemset mining (HAUIM) is a variation in HUIM that selects patterns by considering both their utilities and lengths. In the last decades, several algorithms have been introduced to mine high average-utility itemsets. To speed up the HAUIM process, here, a new algorithm is proposed, which uses a new list structure and pruning strategy. Several experiments performed on the real and synthetic datasets show that the proposed algorithm outperforms the state-of-the-art HAUIM algorithms in terms of runtime and memory consumption.

Keywords: *Data Mining, Frequent Pattern, Utility, High Average-utility Itemset.*

1. Introduction

Nowadays, the quantity of digital data is growing exponentially and finding appropriate data is increasingly challenging. Hence, it is impossible to obtain useful knowledge without computers and data mining techniques. Frequent Itemset Mining (FIM) is one of the most important tasks of data mining introduced by Agrawal [1]. The problem is defined as mining the set of frequent patterns. An itemset X is a frequent pattern, if it appears in a dataset more than a predefined threshold. Many algorithms have been developed to find frequent itemsets [2]–[7]. Most of these algorithms use the downward closure property (Apriori property) to prune search space; according to this property, if a set is frequent, all of its subsets must be frequent as well. Frequent patterns are widely used in many applications such as market basket analysis, web mining, and biological analysis. Since the publication of the original paper, several researchers have tried to extend this idea, and hence many variants of FIM such as temporal pattern mining, sequence pattern mining, and rare pattern mining have been proposed.

Although frequent pattern mining plays a fundamental role in data mining, it assumes that all items have the same utility, and their occurrence in transactions is binary. Therefore, these algorithms cannot find high-utility itemsets in some databases. For example, in a supermarket database, items have different prices/profits. Moreover, in each basket, one item may appear more than once. Frequent pattern mining extracts all frequent patterns; however, some frequent patterns may have a low profit. High Utility Itemset Mining (HUIM) is a generalization of the frequent pattern mining problem, which addresses the above issue [8]–[12]. In this problem, items can appear more than once in a transaction, and they have different utilities. Therefore, there are two concepts of internal (frequency in the transaction) and external utility (the actual utility of the item). The purpose is to find the itemsets that provide a higher utility in the entire database. The utility of an itemset X in a transaction is the sum of products of the internal utility of each item from X and their external utility. Moreover, the utility of an itemset in a whole dataset is the sum of the itemset utilities in all

transactions containing X. All itemsets whose overall utility is greater than the pre-defined threshold are considered as high-utility itemsets. Finding high-utility itemsets is not an easy task, because, contrary to the frequent patterns, the high-utility patterns do not have downward closure property. In other words, a low utility itemset may have a high utility supersets. Most of the proposed algorithms use overestimation of utility to define a new criterion with downward closure property for pruning the search space.

According to the definition of high-utility itemsets, as the itemset is longer, it will have a higher utility. This causes to generate too many long itemsets as high utility patterns. Several methods have been proposed to solve this problem. Most of these algorithms look for high average-utility itemsets instead of finding high-utility itemsets. In other words, they try to eliminate the direct impact of the number of items on the final utility through dividing the itemset utility by its length [13]–[16]. An obvious solution is to generate all high-utility itemsets and then separate the ones with high average utility as output. This method is time-consuming. Therefore, the goal of the HAUIM[1] methods is to use average utility to reduce the search space as well. The primary algorithms work based on Apriori such as [13], [15], [17]. These algorithms require multiple scans of the database, and in each scan, they generate many candidate itemsets, which should be counted in the next scan. Another category is the tree-based algorithms [15], [16], which generate candidate itemsets without multiple database scans. These methods generate conditional trees recursively, which is also time-consuming. The list-based methods try to store the database in the lists and then explore the lists to find high-utility itemsets in a depth first search manner [18]–[20].

In this research work, a new method is presented for discovering high average-utility itemsets. The proposed method uses a new list structure and pruning strategy to increase the efficiency. The major contributions of this paper are as what follow:

- A novel utility list structure is presented, which is more compact than the previous utility lists.
- A new algorithm is proposed that can mine all high average-utility itemsets using a new utility list structure.
- We also propose a tighter upper bound, which provides a more precise estimate of the average utility, and thus it can further reduce the search space.
- For evaluating the proposed method, several experiments have been performed on real datasets.

The rest of the paper is organized as follows. Previous works are reviewed in Section 2. In Section 3, the problem statement and the required preliminaries are described. Section 4 introduces the proposed method. The experiments carried out are analyzed in Section 5, and Section 6 concludes the paper. All symbols are defined in table 1.

Table 1. Nomenclature.

Symbol	*Meaning*
I	List of all Items
T	List of all transactions
m	Number of items
n	Number of transactions
T_q	A transaction
$u(i_j, T_q)$	Utility of the item i_j in T_q
$u(X, T_q)$	Utility of an itemset X in T_q
$u(X)$	Utility of an itemset X in a database
$tu(T_q)$	Utility of T_q
$au(X,T_q)$	Average utility of X in T_q
$au(X)$	The average utility of X in a database
$tmu(T_q)$	The maximum utility of T_q
$AUUB(X)$	Average Utility Upper Bound of X
$tmu(X,T_q)$	maximum average-utility of X in a T_q
A_X	Set of all items a that $\forall\ b \in X, a > b$ and also $u(a,T_q) > au(X,T_q)$

2. Literature survey

High-utility itemset mining is an extension of the frequent pattern-mining problem. As stated in the previous section, high-utility itemsets do not have the downward closure property, and the search space cannot be easily pruned in their mining process, because if an itemset has a utility less than threshold, a superset of it may have a higher utility than the threshold. In other words, a high-utility item may be added to a low-utility itemset, which may produce a high-utility superset. Therefore, in the high-utility itemset mining algorithms, finding a pruning criterion that has the downward closure property is very important. This criterion should be an overestimation of the utility of itemsets so that

[1] High Average Utility Itemsets Miner

pruning does not affect the correctness of the algorithm. Liu et al. introduced the concept of Transaction Weighted Utilization (TWU) for the first time [8]. For each itemset X, TWU is equal to the sum of the total utility of all transactions that contains X. This value is an upper bound, and is always greater than or equal to the actual utility of the itemset X. Therefore, if it is less than the threshold, the utility of the itemset is also less than the threshold. This feature has the downward closure property. In other words, any superset Y of X can only occurs in the same transactions in which X appears. Therefore, the sum of the total utility of transactions in which Y occurs is less than the sum of the utility of transactions in which X occurs. Therefore, if TWU of X is lower than *minUtil*, then TWU of all of its supersets will be also less than *minUtil* and can be pruned. Based on this feature, Liu et al. [8] introduced a two-step algorithm. In the first step, all candidate itemsets whose TWU are not less than the threshold are found based on the Apriori algorithm with several database scans. Then, in the second step, using another scan, the real utility of these itemsets are obtained and high-utility itemsets are discovered.

Although the Apriori and TWU-based methods have reduced the search space greatly, they have to scan the database for several times to generate candidate itemsets. In order to solve this problem, tree-based methods have been introduced. These algorithms first convert the database into a tree form; then, without having to scan the database, they find the candidate itemsets using the data stored in the tree and recursive methods[21]–[23]. In these algorithms, the search space pruning is based on TWU's, i.e. the search space is still large and there are plenty of conditional trees, which is time-consuming. In addition, these methods have to find candidate itemsets that have a TWU greater than the threshold, and then find the real utility of the itemsets with another scan.

List-based solutions attempt to generate high-utility itemsets without the need to generate candidate itemsets [24]–[26]. In these methods, first, all the necessary information from the database is stored in the form of lists in the main memory. Then the search space is traversed in depth first order, and high-utility itemsets are found by extending the patterns, merging the lists, and computing the utility of itemsets using information in the lists. In these algorithms, the memory consumption of the lists and the pruning methods are of great importance.

Since the initial definition of a high-utility itemset does not take into account the length of the itemset, longer itemsets are more likely to be selected as high-utility itemsets. Several algorithms are proposed that try to consider the length of an itemset in the process of high-utility itemset mining. For example, in [27], in addition to the utility of an itemset, there is also a limitation on the length of the output itemsets, that is, the length of high-utility itemsets should be less than the pre-defined minimum length as well. Several other methods are proposed to deal with this problem that use the average utility of an itemset instead of its utility[14], [15], [17]–[20]. In other words, in order to include the length of an itemset in the mining process, the total utility of that itemset is divided by its length. Contrary to the HUIM problem in which it is possible to use TWU for the initial pruning, in high average-utility itemset mining, TWU cannot be used for pruning because the average-utility of an itemset may be less than the average-utility of its subsets. For example, if $T_q = \{a,b\}$, $X = \{b\}$ and the utility of a is greater than the utility of b, then the utility of T_q is equal to the sum of the utilities of a and b that are greater than the utility of X , while the average-utility of T_q will be less than the average-utility of X . Therefore, the transaction maximum utility criterion is used as an upper bound for itemset utility in these algorithms. The maximum utility of each itemset is always greater than the average utility of all its subsets. Therefore, the AUUB (Average Utility Upper Bound) can be used for pruning, which is equal to the sum of the maximum utility of the transactions in which the itemset has occurred.

The first algorithm to solve the high average-utility itemset mining problem is the TPAU method [17], which is based on Apriori. Similar to the two-stage methods, this algorithm first finds candidate itemsets whose average utility upper bound is not less than the pre-defined threshold and then finds the real average-utility with rescanning the database. Multiple database scans and generation of the large number of candidate itemsets are the weaknesses of this method. The HAUI-Growth method is a tree-based method [15], which eliminates the need for multiple scans and generation of candidate itemsets by keeping information about items and their utilities in a tree. The HAUI-Tree [16] is another tree-based method that accesses to the database projection in the main memory by TIDs, and therefore, there will be no need for multiple scans of the database. Some new list-based methods such as HAUI-Miner [18], MHAU [19], and EHAUPM [20] have also been

introduced recently. These methods are explained in more details in the next section.

3. Problem statement and required preliminaries

3.1. Problem statement

Suppose that $D=\{T_1, T_2, \ldots, T_n\}$ is the set of all transactions, and $I=\{i_1, i_2, \ldots, i_m\}$ is the set of all distinct items in D. Each transaction is presented as $T_q=\{i_1, i_2, \ldots, i_l\}$, $T_q \in D$, which has a unique identifier called TID. A k-itemset X ($X=\{i_1, i_2, \ldots, i_k\}$) is a sub-set of I with k items. If an itemset X is a subset of a transaction T_q ($X \subseteq T_q$), then we can say that T_q contains X or X appearing in T_q. The external utility of an item i_j is presented as $pr(i_j)$ that is equal to the profit value of that item. The internal utility of an item i_j in a transaction T_q is represented as $q(i_j, T_q)$ that is equal to the number of item i_j in the transaction T_q. Table 2 shows a sample database containing six transactions and six distinct items with different utilities. Table 3 specifies the external utility for each item.

Table 2. A sample database.

TID	Transaction(item, quantity)
1	A:2, B:3, C:2, D:4
2	C:4, E:5, F:1
3	B:2, C:5, F:6
4	A:5, B:6, C:4, E:7, F:2
5	A:3, B:7, C:4, E:5,F:5
6	D:4, E:6, F:5

Table 3. External utility.

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

According to table 2, the internal utility of the item A in the transaction 1 is equal to 2 and the internal utility of the item E in the transaction 4 is equal to 7. Moreover, according to table 3, the external utility of item A is 4 and the external utility of item E is 2.

The following definitions are required in the rest of this paper:

Definition 1 (the utility of the item i_j): the utility of an item i_j in a transaction T_q is defined as the product of $q(i_j, T_q)$ and $pr(i_j)$, as:

$$u(i_j, T_q)=q(i_j, T_q)\times pr(i_j) \quad (1)$$

For example, in table 2, $u(\mathrm{E},4)=14$ and $u(\mathrm{A},1)=8$.

Definition 2 (the utility of an itemset X in a transaction T_q): the utility of an itemset X in a transaction T_q that is represented as $u(X, T_q)$ is calculated as:

$$u(X, T_q)=\sum_{i_j \in X \wedge X \subseteq T_q} q(i_j, T_q)\times pr(i_j) \quad (2)$$

For example, $u(ABC, T_1)=2\times4+3\times3+2\times1=19$.

Definition 3 (the utility of an itemset X in a database D): the utility of an itemset X in a database is defined as:

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

For example, $u(ABC)=19+42+37=98$.

Definition 4 (utility of the transaction T_q): the utility of a transaction T_q, that is represented as $tu(T_q)$ is the sum of all its item utilities:

$$tu(T_q)=\sum_{i_j \in T_q} u(i_j, T_q) \quad (4)$$

For example, $tu(T_1)=2\times4+3\times3+2\times1+4\times3=31$.

Definition 5 (the total utility of a database D): the total utility of a database is the sum of all its transaction utilities, that is:

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

For example, $TU=31+18+35+64+54+44=246$.

Definition 6 (the average utility of a k-itemset in a transaction T_q): the average utility of an itemset X with k items in a transaction T_q ($au(X, T_q)$) is defined as:

$$au(X, T_q) = \frac{\sum_{i_j \in X \wedge X \subseteq T_q} q(i_j, T_q) \times pr(i_j)}{k} \quad (6)$$

In this formula, k is the number of items in X ($|X| = k$). For example, $au(CD, T_1) = \frac{2\times1+4\times3}{2} = 7$ and $au(ACD, T_1) = \frac{2\times4+2\times1+4\times3}{3}$; 7.33

Definition 7 (the average utility of a k-itemset in a dataset D): this utility, denoted as $au(X)$, is equal to the sum of the average utilities of X in all transactions in D, that is:

$$au(X) = \sum_{X \subseteq T_q \wedge T_q \in D} au(X, T_q) \quad (7)$$

For example, $au(EF) = au(EF, T_2) + au(EF, T_4) + au(EF, T_5) + au(EF, T_6) = 7+11+15+16 = 49$

Definition 8 (HAUIM: High Average-Utility Itemset Mining): the high average-utility itemset mining problem attempts to find all itemsets whose average-utility in the entire database D is not less than the pre-defined threshold (minUtil). This value is obtained as:

$$HAUI = \{X \mid au(X) \geq TU \times \delta\}. \quad (8)$$

where δ is an arbitrary percent value between 0% and 100% predefined by the user.

Definition 9 (Transaction Maximum Utility): previous algorithms for HAUI mining use the maximum utility of each transaction to reduce the search space. The transaction maximum utility of T_q is presented as $tmu(T_q)$, which is equal to the largest utility of the items in the transaction T_q. This value is obtained by the following formula:

$$tmu(T_q) = max(\{u(i_j) \mid i_j \in T_q\}). \quad (9)$$

For example, $tmu(T_2) = max(4, 10, 4) = 10$.

Definition 10 (the Average-Utility Upper Bound of the itemset X): this is denoted as AUUB (X) and is equal to the sum of the maximum utilities of all transactions in which X appears and it is calculated as:

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

For example, $AUUB(E) = tmu(T_2) + tmu(T_4) + tmu(T_5) + tmu(T_6) = 10+20+21+20 = 71$. Table 4 shows AUUBs of all items in our running example.

Table 4. AUUBs.

Item	A	B	C	D	E	F
AUUB	53	77	87	32	71	95

3.2 Preliminaries

In this section, we explain three state-of-the-art HAUIM algorithms in more details.

3.2.1 HAUI-miner

HAUI-Miner [18] uses the AU-List structure to store all the required information. In AU-List, there is one row for each transaction that contains X. Each row has three fields: tid (transaction T_q identifier), iu (real utility of X in the transaction T_q) and mu (maximum utility of items greater than X in the transaction T_q).

HAUI-Miner finds the maximum utility of all transactions at the first scan of a database, and then calculates the AUUB for each item. Then it deletes all the items with AUUB less than the threshold from the database. For example, if the threshold (minUtil) is considered to be 40 in the running example, the data item D is deleted. Then all transactions are sorted according to the order of AUUB. In the current example, the order of the items will be in the form of $A \text{ p } E \text{ p } B \text{ p } C \text{ p } F$. The new database is called the modified database, and is represented as D'. The modified database for the current example is shown in table 5. The number in front of each item in this table is the real utility (the internal utility multiplied by the external utility).

Table 5. Modified database.

TID	Transaction (item, quantity)
1	A:8, B:9,C:2
2	E:10, C:4, F:4
3	B:6, C:5, F:24
4	A:20, E:14, B:18, C:4, F:8
5	A:12, E:10, B:21, C:4, ,F:20
6	E:12, F:20

After constructing the AU-lists for all items, all high average utility itemsets are generated by combining these lists, recursively. Suppose that X is the itemset that has extended the prefix P by adding i_x (or $X = p \cup i_x$, $p = \{i_1, i_2, \ldots, i_p\}$ and i_1 p i_2 p … p i_p p i_x). Then to extend X, it is possible to combine it with any itemset Y ($Y = p \cup i_j$) where i_j f i_x. At each stage, pruning the search space is performed based on AUUB, and the database is projected based on the itemset X. In addition, new lists are obtained by combination of the previous lists.

The disadvantage of this method is that the highest utilities of items in a transaction is considered as the AUUB of all itemsets appearing in that transaction. Although this value is usually much higher than the real average, it can be used as the upper bound in the search space pruning. Hence, this method is not efficient and useful in large databases.

3.2.2 MHAI

MHAI uses the remaining items and the remaining maximum utilities for efficient pruning [19]. Using these new criteria, more candidate itemsets are pruned. Similar to HAUI-Miner, the database is scanned first and AUUB of all items are computed. In the second scanning, transactions are read one by one. First, items that have an AUUB less than the threshold are removed from the transaction, and then the rest of the items are sorted in an AUUB ascending order. For each item, i_j, that remains in the transaction, if its HAI-list has not been created yet, an HAI list is created; otherwise, its previous list will be updated. In this method, the lists are similar to the HAUI-Miner method, with the exception that in each list, in addition to the rows, the itemset X and the maximum remaining items (*mn*) are also stored.

Definition 11 (the maximum remaining items): suppose that a transaction T_q is arranged in an AUUB ascending order and i_x is the last item in X. If $n(i_x, T_q)$ specifies the number of items that are placed in the transaction T_q after i_x, the maximum number of remaining items (*mn*) stored in the HAI-List for the itemset X is equal to the maximum value of $n(i_x, T_q)$ for all transactions T_q where X occurred. It is defined as:

$$mn(X) = max\left(n(i_x, T_q) \mid X \subseteq T_q\right) \quad (11)$$

Definition 12 (remaining maximal utility): suppose that a transaction T_q contains an itemset X. The remaining maximal utility is denoted as $mu(X, T_q)$ and defined by the following formula.

$$mu(X, T_q) = \max(\{u(i_j, T_q) \mid i_j \in T_q \wedge i_j \text{ f } i_x\}) \quad (12)$$

Figure 1 illustrates how these HAI-lists are built for our running example shown in table 5. Each list in this figure is the HAI-list of an itemset. Moreover, in figure 1, sections (a), (b), and (c) show the updated HAI-lists after processing of T_1, T_2 and T_3 respectively. For example, by processing the third transaction and seeing an item B, since the list of B has already been made, it is updated. The utility of B in this transaction is 6, and the maximum utility after the B belongs to F (24); therefore, it adds a row to the previous structure with *TID*, *iu* and *mu* utilities of 3, 6, and 24. The maximum number of items after B in this transaction is 2, so the *mn* value changes to 2.

After generating HAI lists in the MHAI algorithm, the process of building and combining the lists is performed based on the HAUI-Miner method. Accordingly, $list_y$ is combined with $list_x$ and $list_{xy}$ is built. To build $list_{xy}$, for each row *e* shared in $list_x$ and $list_y$, a new row e is built in $list_{xy}$ and its *iu* is equal to $list_x.e.iu + list_y.e.iu - list_p.e.iu$. ($list_p$ is the list of the shared prefix; if the itemset corresponding to $list_x$ is a single member, $list_p.e.iu$ is considered zero). Besides, its *mu* is equal to $list_y.e.mu$ and $list_{xy}.mn = list_y.mn$.

The MHAI method attempts to use pruning before generating larger itemsets to reduce the time and the memory usage. In this method, a new pruning criterion is used, which is stricter than HAUI-Miner pruning criterion and can reduce the search space better. In MHAI, the maximum average utility of itemset X is used.

A	mn:2	
TID	iu	mu
1	8	9

B	mn:1	
TID	iu	mu
1	9	2

C	mn:0	
TID	iu	mu
1	2	0

a) HAI-lists after processing T_1

A	mn:2	
TID	iu	mu
1	8	9

E	mn=2	
TID	iu	mu
2	10	4

B	mn:1	
TID	iu	mu
1	9	2

C	mn=1	
TID	iu	mu
1	2	0
2	4	4

F	mn=0	
TID	iu	mu
2	4	5

b) HAI-lists after processing T_2

A	mn:2	
TID	iu	mu
1	8	9

E	mn=2	
TID	iu	mu
2	10	4

B	mn:2	
TID	iu	mu
1	9	2
3	6	24

C	mn=1	
TID	iu	mu
1	2	0
2	4	4
3	5	24

F	mn=0	
TID	iu	mu
2	4	0
3	24	0

c) HAI-lists after processing T_3

Figure 1-AU-lists after processing T_1, T_2 and T_3.

Definition 13 (maximum average-utility in a transaction): the maximum average-utility of an itemset X in a transaction T_q is denoted as $mau(X, T_q)$, and it is calculated as:

$$mau(X, T_q) \quad (13)$$

$$\begin{cases} \dfrac{u(X,T_q)+mu(X,T_q)\times mn(X)}{|X|+mn(X)}, & mu(X,T_q) > \text{au}(X,T_q) \\ \dfrac{u(X,T_q)+mu(X,T_q)}{|X|+1}, & 0 < mu(X,T_q) \le \text{au}(X,T_q) \\ 0 & mu(X,T_q) = 0 \end{cases}$$

Definition 14 (maximum average-utility of an itemset X In a database D): the maximum average-utility of an itemset X in a database D is shown by $mau(X)$ and obtained as:

$$mau(x) = \sum_{X \subseteq T_q \in D} mau(X, T_q) \quad (14)$$

This criterion shows the maximum utility of any superset built based on X. Therefore, if this value is less than the threshold, the X extension can be excluded because any set built based on X cannot have a high average-utility.

The recursive procedure of high-average utility itemset mining in MHAI first selects a $list_x$ from the lists and calculates the average utility of X, which is equal to the sum of the average utility of itemset X in all transactions that contain X. If the average utility is not less than the threshold, it will go to the output as a high average utility itemset. Then, the maximum average-utility is calculated for this itemset. If it is less than the threshold, the recursive process backs to the previous step; otherwise, it extends the itemset X and generates new lists based on it.

One of the weaknesses of this method is the unrealistic upper bound considered for the average-utility of itemsets. In other words, due to overestimation, the search space is not well-pruned, which increases the memory consumption and runtime.

3.2.3 EHAUPM algorithm

The EHAUPM [20] algorithm addresses the MHAI problem by defining the optimal upper bound for the average-utility as well as the use of various strategies in pruning the search space. There are three ways to reduce the search space in this algorithm. Initially, like similar algorithms, all items whose AUUB is less than the threshold are deleted from the database. In the second step, the AUUB is calculated for all pairs of items in the database and stored in the EAUCM matrix. It can be concluded that if AUUB of each pair of items is less than the threshold, the average utility of the whole itemset that is made with these two items is also less than the threshold and will prevent them from expanding. Moreover, EHMUPM uses two tighter upper-bound models (the Looser Upper-Bound (*lub*) and the revised tighter upper-bound (*rtub*)) to further reduce the search space for mining HAUIs.

One of the weaknesses of this method is that it constructs the co-occurrence matrix and calculates

AUUB for each pair of items, which affects the speed and the amount of memory used for large databases.

4. Proposed algorithm

The HAUI_Miner algorithm uses the projected database. Although the authors' goal is to reduce the search space and runtime of the algorithm, constructing the projected database itself is time-consuming, If the database is projected for each item in the mining process, the database has to be scanned for several times, which is very time-consuming. In addition, if all sub-databases are created in the first scan, the algorithm requires a lot of memory to save them. In contrast to HAUI-Miner, the MHAI method, initially removes all items with low AUUB from the database and then stores all the required information in a list structure. Several experiments in the corresponding paper show that MHAI occupies less memory than HAUI-Miner. Moreover, this algorithm uses a tighter criterion for pruning, which makes the search space more pruned, and thus has a lower running time than HAUI-Miner.

In the proposed algorithm, a new structure for storing information and a new method for pruning the search space are introduced, which would increase the efficiency.

4.1. Proposed structure

In the proposed algorithm, similar to the previous methods, after AUUB calculation for all items, those items with AUUB less than the threshold are removed from the database. Then, the required information of the database is stored in a TID list structure and a set of transactions.

For each item, the TID list stores the TID of all transactions containing that data item. This structure for our running example is shown in figure 2. In contrast to the AU-list structure in the previous approaches, we do not save the item utility and remaining maximal utility in this structure. Hence, memory consumption of each row in the proposed structure is one-third of that in the previous methods.

Instead of storing in AU-lists, we store some necessary information in a transaction list, i.e. all transactions are stored in another separate list. There is one element for each transaction that contains two arrays (one array for the transaction items and another for their corresponding utilities). Figure 3 shows this structure. This transaction list is also used for pruning.

A	E	B	C	F
1	2	1	1	2
4	4	3	2	3
5	5	4	3	4
	6	5	4	5
			5	6

Figure 2. TID lists for modified database in Table 5.

Since for each candidate itemset we should create an AU-List, the memory consumption for AU-lists is proportional to the number of candidate itemsets ($O(\#Candidate\ Itemset)$). Hence, tighter pruning causes to generation of less candidate itemsets and lists, which leads to less memory usage. As a result, pruning unpromising candidate itemsets can compensate for the transaction list overhead in the proposed method. Moreover, EHAUIM stores AUUB for all pair items in the ECAUPS structure, which needs $o(m^2)$ memory space (where m is the number of distinct items in the dataset).

{A,B,C}	*{E,C,F}*	*{B,C,F}*	*{A,E,B,C,F}*	*{A,E,B,C,F}*	*{E,F}*
{8,9,2}	*{10,4,4}*	*{6,5,24}*	*{20,14,18,4,8}*	*{12,10,21,4,20}*	*{12,20}*

Figure 3. The array structure for storing all transactions.

4.2. Mining process

The search space is represented as an enumeration tree. Each node is an itemset that may be a high average utility itemset. The proposed algorithm scans the search space using a depth-first search. At each step, the itemset X is extended by adding an item (whose AUUB value is greater than the AUUB of all items appearing in X). As an itemset expands, the average-utility of the extended itemset is calculated. If it exceeds the pre-defined threshold, it is added to the HAUI list. Suppose that X is an itemset that has extended the prefix P by adding i_x (or $p = \{i_1, i_2, \ldots, i_p\}$, $X = p \cup i_x$ and i_1 p i_2 p … p i_p p i_x); Then, we can extend X by combining it with a set Y ($Y = p \cup i_y$ that i_y f i_x) to construct a new candidate itemset XY ($XY = p \cup i_x \cup i_y$). The TID lists of XY is achieved by intersection of the TID lists of X and Y. The average-utility of XY is calculated using this list and the information stored in the transaction set. In other words, it is possible to retrieve the utility of the items and calculate the average utility by referring directly to the transactions specified in the TID list.

As mentioned earlier, since the downward closure property does not hold for the average-utility value, an itemset might have a low average utility value but one of its extended might has a high average value. Therefore, if during the exploration of the search space a low utility itemset appears, it should be extended and cannot be pruned. Due to the large size of the search space, finding an appropriate way to prune it is necessary. The HAUI-Miner method uses the transaction maximum utility for pruning. This overestimation cannot prune the search space well. In the MHAU method, instead of the transaction maximum utility, the remaining maximum utility is used, which is closer to reality. However, for the calculation of the maximum average-utility, the number of remaining items is also required. To save the memory, instead of keeping the number of remaining items for each transaction containing X, the largest one is stored and calculations are made based on it. This also leads to having looser upper bound.

In the proposed method, TID list of an itemset X stores all TIDs of transactions that contain X. Since all the required information about transactions is stored in the transaction set, we can access to them using their TID. Hence, the maximum average utility for any supersets of X can be estimated more precisely because the utility of all remaining items are available.

Definition 15 (the maximum average utility of an itemset X in a transaction T_q): as mentioned earlier, $u(X)$ is the real utility of X in the entire database and $u(X,T_q)$ is the real utility of X in the transaction T_q; assume that A_X is the set of all items that are larger than all items of X in AUUB order and also have a utility value greater than the average utility of X, that is:

$$A_X = \{a \mid a \in T_q \ \&\forall\ b \in X \wedge a > b \wedge u(a,T_q) > au(X,T_q)\} \quad (15)$$

The maximum average utility of an itemset X in a transaction T_q is denoted as $mau(X,\ T_q)$, and is calculated as:

$$mau(X,\ T_q) = max_{A'_X \subseteq A_X}\left(\frac{u(X,T_q)+\sum_{a\in A'_X} u(a,T_q)}{|X|+|A'_X|}\right) \quad (16)$$

To achieve A'_X, A_X is arranged in the ascending order of utility values in T_q and then items of A_X are added to A'_X from the largest utility to the least until the average utility decreases.

Definition 16 (maximum average-utility of an itemset X in a database D): similar to formula 14, if the $mau(X,\ T_q)$ value for all transactions that contain X is summed-up, the maximum average-utility of an itemset X is obtained in the entire database.

Theorem 1: $mau(X,\ T_q)$ is greater than or equal to $au(X,\ T_q)$ (average utility of X).

PROOF:

$$mau(X,\ T_q) = \frac{u(X,\ T_q)+\sum_{a\in A'_X} u(a,\ T_q)}{|X|+|A'_X|} \geq \frac{u(X,\ T_q)+\sum_{a\in A'_X} \frac{u(X,\ T_q)}{|X|}}{|X|+|A'_X|} = \frac{u(X,\ T_q)+|A'_X|\times\frac{u(X,\ T_q)}{|X|}}{|X|+|A'_X|}$$

$$= \frac{u(X,\ T_q)\times(|X|+|A'_X|)}{|X|\times(|X|+|A'_X|)} = au(X,T_q)$$

$$\Rightarrow mau(X,\ T_q) \geq au(X,\ T_q) \ \blacksquare$$

Theorem 2: MAU has downward closure property

PROOF*:* Suppose that $X \cup z$ is one of the X supersets. If *mau* has a downward closure property, $mau(X,\ T_q)$ should be greater than or equal to the *mau* of all of its supersets ($mau(X \cup z,\ T_q) \leq mau(X,\ T_q)$. Based on the utility of the new added item (z), two modes happen:

First mode: the utility of z in T_q is equal to or greater than the average utility of X in T_q; in other words, $u(z,T_q) \geq au(X,T_q)$. Accordingly, $z \in A_X$ and the average utility of $X \cup z$ is greater than or equal to $au(X,T_q)$. Hence, $A_{X\cup z} \cup z \subseteq A_X$, thus $mau(X \cup z) \leq mau(X)$

Second mode: the utility of the element z in T_q is less than the average utility of X; in other words, $u(z,T_q) < \mathrm{au}(\mathrm{X},T_q)$. In this case, $A_{\mathrm{X}\cup\mathrm{z}}$ is defined as:

$$A_{\mathrm{X}\cup\mathrm{z}} = \{a \mid a \in A_X \ \&\forall\ , a\rangle z\} \cup \{a \mid a \in X \ \& a\rangle z \ \&u(a,T_q) < au(X,T_q)\}$$

Suppose that M' and N' are sub-sets of M and N, for which the maximum utility of $A_{\mathrm{X}\cup\mathrm{z}}$ is achieved. The average utility of the items of N' is less than $au(X,T_q)$ and the average utility of the items of M' is greater than $au(X,T_q)$.

$$mau(X\cup \mathrm{z},T_q) = \frac{u(X,T_q)+\sum_{a\in M'}u(a,T_q)+u(z,T_q)+\sum_{a\in N'}u(a,T_q)}{|X|+|M'|+1+|N'|} \le \frac{E+(|N'|+1)*\frac{E}{F}}{F+|N'|+1} = \frac{E}{F} = mau(X,T_q)$$

Since X is a sub-set of $\mathrm{X}\cup\mathrm{z}$, the transactions containing $\mathrm{X}\cup\mathrm{z}$ are subsets of transactions that contain X. Moreover, *mau* is a positive value; hence:

$$mau(\mathrm{X}\cup\mathrm{z}) = \sum_{\mathrm{X}\cup\mathrm{z}\subseteq T_q\in D} mau(\mathrm{X}\cup\mathrm{z},T_q) \le \sum_{\mathrm{X}\cup\mathrm{z}\subseteq T_q\in D} mau(X,T_q) \le \sum_{X\subseteq T_q\in D} mau(X,T_q) = mau(X)$$

$$\Rightarrow mau(\mathrm{X}\cup\mathrm{z}) \le mau(X) \ \blacksquare$$

At the mining step, all itemsets are expanded according to the search space shown in figure 4. As the X set is extended, the transaction data is accessed based on the TID list and its real average-utility is obtained. If this value is greater than the threshold, X is added to the list of high average-utility itemset. Then, *mau* is calculated for all itemsets, whether they have a high utility or not. If $mau(X)$ is less than the threshold, since $mau(X)$ is the upper bound of average-utility that a superset X can have, it is not possible to find a superset in the search space whose average utility is more than the threshold. Therefore, the subtree of the X itemset in the search space is pruned. Reducing search space generates fewer candidate itemsets and thus reduces the memory and time required to create TIDs and calculate the real average-utility.

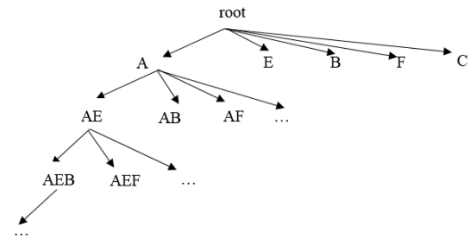

Figure 4. Search space base on the modified database in Table 5.

Since in the new pruning method uses the real value of utilities instead of using the maximum utility, the estimation is closer to reality, which makes the search space more pruned than the previous methods.

5. Evaluation

In this section, the proposed method has been compared in terms of *runtime*, *memory consumption* and *candidate count* with two state-of-the-art algorithms (EHAUPM and MHAI algorithms [17], [18]).

The code for the EHAUPM algorithm is taken from the SPMF package [28]. In this package, more than 150 algorithms are implemented in the field of data mining, frequent pattern mining, and association rules in Java. We have implemented the MHAI algorithm and the proposed algorithm in Java as well. All experiments are carried out on a computer with Windows 10, Intel Corei3, 2.40GHz, 4GB RAM. In order to compare the performance of the proposed algorithm with the existing algorithms, three real datasets and 5 synthetic datasets are used. Information about these datasets is presented in table 6. It should be noted that the execution of the EHAUPM algorithm on the *Connect* database had *out of memory* error, and information about this run is not provided in the comparisons.

Table 6. Parameter of the databases.

Database	Transaction Count	Unique Item Count	Average Transaction Length
accident	340183	468	33.8
Chess	3196	75	37
Connect	67557	129	43
T35 I100D|X|K	X×1000 (x=1 to 5)	100	35

5.1. Memory usage

In this sub-section we investigate memory consumption of the algorithms (the proposed

algorithm, EHAUPM, and MHAI). The peak memory consumption of the algorithms has been measured by utilizing the standard Java API. The results obtained are shown in figure 5 and table 7. Accordingly, with increasing *minUtil* ($TU \times \delta$), memory consumption decreases. The reason is that by increasing the *minUtil*, the minimum average-utility will also increase and the search space is likely to be reduced due to the more pruning. Thus less memory space is required.

As it can be seen in figure 5, the memory consumption in the proposed algorithm is far less than the other algorithms. In comparison to EHAUPM and MHAI, the proposed method can reduce memory consumption by up to 10 and 6 times, respectively. This reduction has two reasons; firstly, the proposed method optimizes the list structure, and secondly, it chooses a better strategy for pruning, which reduces memory consumption. Moreover, although EHAUPM, in some situation, can prune more candidate itemsets, storing ECAUPS structure causes a memory overhead of $o\left(m^2\right)$.

5.2. Runtime

Figure 5 and table 7 show the runtime of the algorithms for different minimum average utility thresholds. Note that, runtime, here, means the total time used for running the algorithms, which is the period between input and output. As the minimum average utility increases, the number of itemsets whose utilities are greater that minimum threshold are decreased and runtime decreases as well.

According to figure 5, the performance of the proposed algorithm is far better than EHAUPM and MHAI. More specifically, it is 5-14 times faster than MHAI and 15-367 times faster than EHAUPM. The reason is that our proposed approach can prune more unpromising candidate itemsets compared to EHAUPM and MHAI, and it does not require to perform the costly join operation for creating the supersets of these pruned candidate itemsets. MHAI uses the maximum remaining items and remaining maximal utility to estimate maximum average remaining utility of an itemset X. It assumes that all transactions approximately have the same length and it stores only one mn value for X (maximum number of items that appear after the last item of X in all transactions containing X). Also assumes that the item utilities in a transaction approximately are the same and only stores the remaining maximal utility for each transactions in the AU-lists. Since the maximum average utility for X is calculated based on these values, the difference between the estimated and actual value, may become high; especially when the difference in length of transactions or item utilities is high, the MAU estimation will become very high, which causes to generate many candidate itemsets. Furthermore, EHAUPM needs to construct ECAUPS (Estimated Average-Utility Co-Occurrence Matrix) that stores the AUUB values of all 2-itemsets. If l is the average length of each transaction, there will be l^2 pair of items in each transaction whose utilities should be evaluated for calculating the *auub* value. Hence, the time complexity overhead of ECAUPS construction is $O\left(l^2 \times n\right)$; in consequence, it is expected that, in comparison to the MHAI and the proposed approach, EHAUPM does not perform well in dense and large datasets.

5.3. Number of candidate itemsets

As described earlier, in each step of the HAUIM algorithms, the search space is explored and the candidate itemsets is generated. For each candidate itemsets, AU-list structure should be constructed to verify that the candidate itemset is HAUI or not.

Accordingly, as the number of candidate itemsets increases, the algorithm slows down and its memory consumption increases. Hence, it is an important factor for comparing HAUI mining algorithms. The number of generated candidates is displayed in figure 5 and table 7. Due to the tighter pruning strategies, the proposed algorithm generates much less candidate itemsets than the other two algorithms.

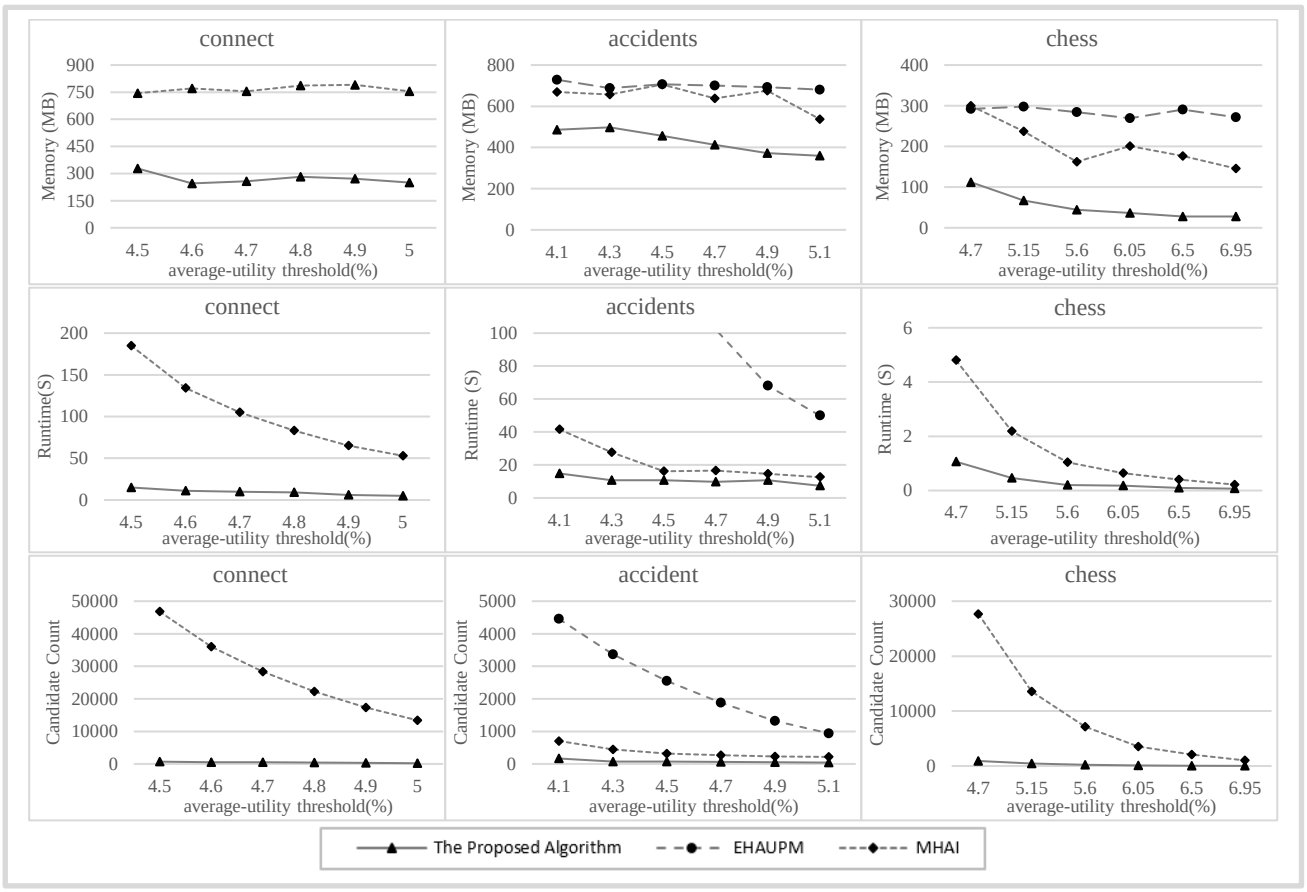

Figure 5. Memory consumption, runtime, and candidate count for various average utility thresholds.

Table 7. Memory consumption, runtime, and candidate count for various average utility thresholds.

		Connect			Accidents				Chess		
	δ	The Proposed Algorithm	MHAI	δ	The Proposed Algorithm	EHAUPM	MHAI	δ	The Proposed Algorithm	EHAUPM	MHAI
Memory (MB)	0.045	332	747	0.041	486	729	669	0.047	112	292	300
	0.046	258	762	0.043	498	687	656	0.051	67	298	237
	0.047	266	775	0.045	457	707	705	0.056	44	284	162
	0.048	289	753	0.047	413	701	638	0.060	36	269	201
	0.049	255	762	0.049	372	693	675	0.065	28	291	176
	0.050	255	748	0.051	360	681	537	0.069	112	292	300
Runtime (S)	0.045	14.5	192.3	0.041	14.4	207.2	41.5	0.047	1	288.5	4.8
	0.046	10.5	139.8	0.043	10.6	164.6	27.6	0.051	0.4	150.6	2.1
	0.047	10.5	109.2	0.045	10.6	128	16.1	0.056	0.2	75.2	1
	0.048	8.8	86.6	0.047	9.7	102.2	16.5	0.060	0.1	40.5	0.6
	0.049	6.8	68.6	0.049	10.6	68.1	14.6	0.065	0.09	20.7	0.4
	0.050	5	51.7	0.051	7.3	49.9	12.5	0.069	0.06	10.8	0.2
Candidate Count	0.045	693	46810	0.041	167	4456	705	0.047	919	1214999	27691
	0.046	571	36048	0.043	72	3363	449	0.051	468	604849	13564
	0.047	514	28405	0.045	72	2554	322	0.056	203	304258	7172
	0.048	435	22300	0.047	60	1882	266	0.060	98	152331	3572
	0.049	314	17327	0.049	54	1320	225	0.065	43	75924	2061
	0.050	215	13406	0.051	41	941	215	0.069	25	37833	1018

5.4. Scalability assessment

In this section, we compare the scalability of the compared algorithms in terms of runtime, memory usage, and candidate itemset counts. The algorithms are performed on several synthetic datasets. For creating the synthetic datasets, we use the database generator tool [28]. We generate 5 datasets with 100 items. Each transaction has up to 50 items, and the number of transactions varies from 100,000 to 500,000 transactions. We indicate this datasets by T35I100D|X|K, in which X varies from 100 to 500 using an increment of 100(K) transactions. The minimum high average threshold is set to $TU \times 0.01$ for all datasets. Figure 6 shows

the runtime and memory consumption of the compared algorithms on the five synthetic datasets mentioned above. Besides, for a better illustration, the candidate itemset counts are represented in the bar graph format in figure 7. As shown in this figure, since the data distribution is constant in these datasets, the number of high average utility and also the number of candidate itemsets are approximately equal in all datasets regardless of dataset size. The results obtained show that the proposed method generates much less candidate itemsets than EHAUPM and MHAI.

Although the number of AU-Lists (number of candidate itemset counts) are approximately equal in all datasets but the size of AU-Lists is different because as the number of dataset transactions increases, the number of entries in each AU-list also increases, and accordingly, the time required for joint operations becomes higher. This causes a significant reduction in the runtime of the proposed method in comparison with the other approaches due to the much lower candidate itemset count (which is illustrated in Figure 6).

Furthermore, figure 6 reveals that the memory usage of EHAUPM and AHIM is raised as the number of transactions is increased. However, since our proposed approach has a more compact AU-Lists (it only store TIDs), dataset size does not influence much in its memory usage. Accordingly, we can determine that the proposed algorithm has the highest scalability performance among the compared algorithms.

Moreover, as shown in figure 5, the proposed algorithm runtime for real datasets increases linearly by decreasing the minimum average utility threshold. However, it is non-linear in other algorithms, which confirms the scalability of the proposed algorithm.

6. Conclusion

In this paper, a new list-based algorithm has been proposed for high average-utility itemset mining task. In the proposed algorithm, an optimal structure was introduced to store the lists. Decreasing of the list size will reduce the memory consumption of the proposed algorithm. Besides, a new strategy has been proposed to prune the search space. Tighter pruning causes generation of less candidate itemsets and lists, which reduces the memory usage and increases the speed of the algorithm. Several experiments have been carried out to evaluate the proposed algorithm on real and synthetic datasets. The results obtained show that the proposed algorithm is more efficient than the state-of-the-art algorithms in terms of time, memory, and number of generated candidate itemsets; more specifically, in the performed experiments, the proposed method is 5-14 times faster than MHAI and 15-367 times faster than EHAUPM. Moreover, the peak memory usage of the proposed method is reduced by up to 10 times.

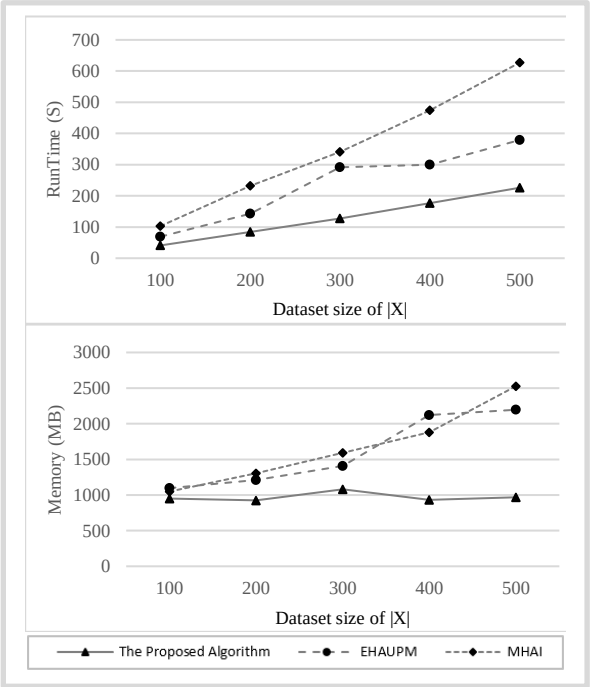

Figure 6. Runtime and memory consumption for various dataset sizes on T35 I100D |X|K.

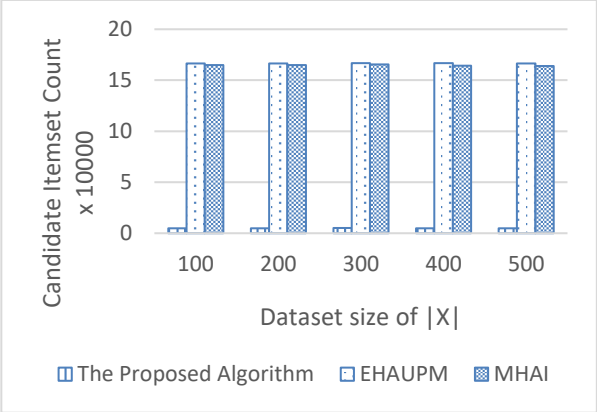

Figure 7. Candidate itemset counts for various dataset sizes on T35 I100D |X|K.

References

[1] Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules. Proceedings of the 20th international conference on very large data bases, VLDB, 1215, pp. 487–499.

[2] Han, J., Pei, J., Yin, Y., & Mao, R. (2004). Mining frequent patterns without candidate generation: A frequent-pattern tree approach. Data mining and knowledge discovery, vol. 8(1), pp. 53–87.

[3] Zaki, M. J., & Gouda, K. (2003). Fast vertical mining using diffsets, Proceedings of the 9th ACM SIGKDD, pp. 326–335.

[4] Borgelt, C. (2005). Keeping things simple: finding frequent item sets by recursive elimination, Proceedings of the 1st international workshop on open source data mining, pp. 66–70.

[5] Pei, J., Han, J., Lu, H., Nishio, S., Tang, S., & Yang, D. (2007). H-mine: Fast and space-preserving frequent pattern mining in large databases. IIE Transactions, vol. 39, no. 6, pp. 593–605.

[6] Deng, Z., Wang, Z., & Jiang, J. (2012). A new algorithm for fast mining frequent itemsets using n-lists, Science China Information Sciences, vol. *55*, *no.* 9, pp. 2008–2030.

[7] Sakenian Dehkordi, M., & Naderi Dehkordi, M. (2016). Introducing an algorithm for use to hide sensitive association rules through perturb technique. Journal of AI and Data Mining, vol. 4, no, 2, pp. 219–227.

[8] Liu, Y., Liao, W.-k., & Choudhary, A. N. (2005). A two-phase algorithm for fast discovery of high utility itemsets, *PAKDD*, vol. *3518*, pp. 689–695.

[9] Li, H.-F., Huang, H.-Y., Chen, Y.-C., Liu, Y.-J., & Lee, S.-Y. (2008). Fast and memory efficient mining of high utility itemsets in data streams. Eighth IEEE International Conference on Data Mining (ICDM'08), pp. 881–886.

[10] Tseng, V. S., Shie, B.-E., Wu, C.-W., & Philip, S. Y. (2013). Efficient algorithms for mining high utility itemsets from transactional databases, IEEE transactions on knowledge and data engineering, vol. 25, no. 8, pp. 1772–1786.

[11] Song, W., Liu, Y., & Li, J. (2014). BAHUI: fast and memory efficient mining of high utility itemsets based on bitmap. International Journal of Data Warehousing and Mining (IJDWM), vol. 10, no. 1, pp. 1–15.

[12] Lan, G.-C., Hong, T.-P., & Tseng, V. S. (2014). An efficient projection-based indexing approach for mining high utility itemsets, Knowledge and information systems, vol. 38, no.1, pp. 85–107.

[13] Hong, T.-P., Lee, C.-H., & Wang, S.-L. (2009). Mining high average-utility itemsets, IEEE International Conference on Systems, Man and Cybernetics, SMC 2009, pp. 2526–2530.

[14] Lan, G.-C., Hong, T.-P., Tseng, V. S., & others. (2012). A projection-based approach for discovering high average-utility itemsets, Journal of Information science and Engineering, vol. 28, no. 1, pp. 193–209.

[15] Lin, C.-W., Hong, T.-P., & Lu, W.-H. (2010). Efficiently Mining High Average Utility Itemsets with a Tree Structure, In Proceedings of Intelligent Information and Database Systems, pp. 131–139.

[16] Lu, T., Vo, B., Nguyen, H. T., & Hong, T.-P. (2014). A new method for mining high average utility itemsets, IFIP International Conference on Computer Information Systems and Industrial Management, pp. 33–42.

[17] Hong, T.-P., Lee, C.-H., & Wang, S.-L. (2011). Effective utility mining with the measure of average utility, Expert Systems with Applications, vol. 38, no. 7, pp. 8259–8265.

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

[19] Yun, U., & Kim, D. (2017). Mining of high average-utility itemsets using novel list structure and pruning strategy, Future Generation Computer Systems, vol. 68, pp. 346–360.

[20] Lin, J. C.-W., Ren, S., Fournier-Viger, P., & Hong, T.-P. (2017). EHAUPM: Efficient High Average-Utility Pattern Mining With Tighter Upper Bounds, *IEEE Access*, vol. 5, pp. 12927–12940.

[21] Tseng, V. S., Wu, C.-W., Shie, B.-E., & Yu, P. S. (2010). UP-Growth: an efficient algorithm for high utility itemset mining," in Proceedings of the 16th ACM SIGKDD, international conference on Knowledge discovery and data mining, 2010, pp. 253–262.

[22] Lin, C.-W., Hong, T.-P., & Lu, W.-H. (2011). "An effective tree structure for mining high utility itemsets," Expert Systems with Applications, vol. 38, no. 6, pp. 7419–7424.

[23] Lin, J. C.-W., Gan, W., Hong, T.-P., & Tseng, V. S. (2015). Efficient algorithms for mining up-to-date high-utility patterns, Advanced Engineering Informatics, vol. 29, no. 3, pp. 648–661.

[24] Liu, M., & Qu, J. (2012). "Mining high utility itemsets without candidate generation," in Proceedings of the 21st ACM international conference on Information and knowledge management, 2012, pp. 55–64.

[25] Fournier-Viger, P., Wu, C.-W., Zida, S., & Tseng, V. S. (2014). FHM: faster high-utility itemset mining using estimated utility co-occurrence pruning, in International symposium on methodologies for intelligent systems, pp. 83–92.

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

[27] Fournier-Viger, P., Lin, J. C.-W., Duong, Q.-H., & Dam, T.-L. (2016). FHM+: faster high-utility itemset mining using length upper-bound reduction, in International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, 2016, pp. 115–127.

[28] Fournier-Viger, P., Gomariz, A., Gueniche, T., Soltani, A., Wu, C.-W., & Tseng, V. S. (2014). SPMF: a java open-source pattern mining library, The Journal of Machine Learning Research, vol. 15, no. 1, pp. 3389–3393.

نشریه هوش مصنوعی و داده کاوی

روشی جدید جهت استخراج مجموعه اقلام با متوسط ارزش بالا

آزاده سلطانی[1] و محمود سلطانی[2]

[1] گروه مهندسی کامپیوتر، دانشگاه بجنورد، بجنورد، ایران.

[2] گروه مهندسی کامپیوتر ، دانشگاه صنعتی قوچان، قوچان، ایران.

ارسال ۲۰۱۸/۰۹/۲۲؛ بازنگری ۲۰۱۹/۰۱/۲۲؛ پذیرش ۲۰۱۹/۰۴/۰۷

چکیده:

مسئله استخراج مجموعه اقلام با ارزش بالا (HUIM)، شاخه‌ی جدیدی از داده‌کاوی است که به علت داشتن کاربردهای فراوان در حوزه‌های مختلف مورد توجه بسیاری از محققان قرار گرفته است. هدف این مسئله، یافتن تمام مجموعه اقلامی است که ارزش آنها از حداقل آستانه مورد نظر کمتر نباشد. در مسئله HUIM ، برای محاسبه ارزش، طول مجموعه‌ها در نظر گرفته نمی‌شود. از آنجاییکه ارزش مجموعه‌هایی با طول بیشتر بالاتر خواهد بود؛ الگوریتمهای HUIM تعداد زیادی الگوی با ارزش، با طول بالا تولید خواهند کرد. مسئله استخراج مجموعه اقلام با *متوسط ارزش بالا* (HAUIM)، ورژن جدیدی از HUIM است که طول مجموعه‌ها را نیز در محاسبه ارزش آنها در نظر می‌گیرد. الگوریتمهای متعددی برای حل این مسئله ارائه شده است. در این تحقیق، برای سرعت بخشیدن به پروسه‌ی کشف مجموعه اقلام با *متوسط ارزش بالا*، الگوریتم جدیدی ارائه شده است که از ساختار داده و استراتژی هرس جدیدی استفاده می‌کند. آزمایشهای متعددی که بر روی مجموعه داده‌های واقعی و مصنوعی انجام شده، نشان می‌دهد که روش پیشنهادی نسبت به روشهای فعلی HAUIM از نظر زمان اجرا و حافظه مصرفی بهتر عمل کرده است.

کلمات کلیدی: داده‌کاوی، الگوی پرتکرار، الگوی با ارزش بالا، الگوی با ارزش متوسط بالا.