

Efficient Mining of High Average-Utility Itemsets with Multiple Minimum Thresholds

Jerry Chun-Wei Lin¹(✉), Ting Li¹, Philippe Fournier-Viger²,
Tzung-Pei Hong^{3,4}, and Ja-Hwung Su⁵

¹ School of Computer Science and Technology,
Harbin Institute of Technology Shenzhen Graduate School, Shenzhen, China
jerrylin@ieee.org, tingli@ikelab.net

² School of Natural Sciences and Humanities,
Harbin Institute of Technology Shenzhen Graduate School, Shenzhen, China
philfv@hitsz.edu.cn

³ Department of Computer Science and Information Engineering,
National University of Kaohsiung, Kaohsiung, Taiwan
tphong@nuk.edu.tw

⁴ Department of Computer Science and Engineering,
National Sun Yat-sen University, Kaohsiung, Taiwan

⁵ Department of Information Management, Cheng Shiu University,
Kaohsiung, Taiwan
bb0820@ms22.hinet.net

Abstract. High average-utility itemsets mining (HAUIM) is a key data mining task, which aims at discovering high average-utility itemsets (HAUIs) by taking itemset length into account in transactional databases. Most of these algorithms only consider a single minimum utility threshold for identifying the HAUIs. In this paper, we address this issue by introducing the task of mining HAUIs with multiple minimum average-utility thresholds (HAUIM-MMAU), where the user may assign a distinct minimum average-utility threshold to each item or itemset. Two efficient IEUCP and PBCS strategies are designed to further reduce the search space of the enumeration tree, and thus speed up the discovery of HAUIs when considering multiple minimum average utility thresholds. Extensive experiments carried on both real-life and synthetic databases show that the proposed approaches can efficiently discover the complete set of HAUIs when considering multiple minimum average-utility thresholds.

Keywords: High average-utility itemsets · Multiple thresholds · Data mining · Downward closure · Utility

1 Introduction

The main purpose of knowledge discovery in database (KDD) is to discover implicit and useful information in a collection of data. Association-rule mining (ARM) or frequent itemset mining (FIM) plays an important topic in KDD,

which has been extensively studied [1, 3]. A major limitation of traditional ARM and FIM is that they focus on mining association rules or frequent itemsets in binary databases, and treat all items as having the same importance without considering factors. To address this limitation, the problem of high utility itemset mining (HUIM) [4, 10, 18, 19] was introduced. An important limitation of traditional HUIM is that the utility of an itemset is generally smaller than the utility of its supersets. Hence, traditional HUIM tends to be biased toward finding itemsets of greater length (containing many items), as these latter are more likely to be high utility itemsets. The utility measure used in traditional HUIM thus does not provide a fair measurement of the utility of itemsets.

To alleviate the influence of an itemset's length on its utility, and find more useful high utility itemset for recommendation, Hong et al. [7] proposed the average utility measure, and the problem of high average utility itemset mining (HAUIM). The average utility of an itemset is defined as the total utility of its items in transactions where the itemset appears, divided by the number of items in the itemset. Numerous algorithms have been designed to more efficiently mine high average-utility itemsets (HAUIs) [11, 13, 14, 16] but most of them rely on a single minimum average-utility threshold to mine HAUIs. In real-life situations, each item or itemset may be more or less important to the user. It is thus unfair to measure the utility of all items in a database using the same minimum utility threshold.

To address this issue, this paper proposes a novel framework for high average-utility itemset mining with multiple minimum average-utility thresholds (HAUIM-MMAU). Based on the proposed framework, a two-phase algorithm named HAUI-MMAU is proposed to discover HAUIs. To improve the performance of the proposed algorithm, two efficient pruning strategies called IEUCP and PBCS are designed to prune unpromising itemsets early, thus reducing the search space and speeding up the discovery of HAUIs. Extensive experiments were conducted on both real-life and synthetic datasets to show that the proposed algorithm can efficiently mine the complete and correct set of HAUIs in databases, while considering multiple minimum average-utility thresholds to assess the utility of itemsets.

2 Related Work

In recent years, HUIM [4, 10, 18, 19] has become a key research topic in the field of data mining. Chan et al. [4] presented a framework to mine the top- k closed utility patterns based on business objectives. Yao et al. [18, 19] defined the problem of utility mining while considering both purchase quantities of items in transactions (internal utility) and their unit profits (external utility). Liu et al. [10] introduced the transaction-weighted utility (TWU) model and the transaction-weighted downward closure (TWDC) property. Lin et al. [11] adopted the TWU model to design the high-utility pattern (HUP)-tree for mining HUIs using a condensed tree structure called HUP-tree. Liu and Qu [15] proposed the HUI-Miner algorithm to discover HUIs without generating candidates using a designed

utility-list structure. Fournier-Viger et al. [5] then presented the FHM algorithm and the Estimated Utility Co-occurrence Structure (EUCS) to mine HUIs.

In HUIM, the utility of an itemset is defined as the sum of the utilities of its items, in transactions where the itemset appears. An important drawback of this definition is that it does not consider the length of the itemset. Hong et al. [7] first proposed the average utility measure, and the stated the problem of HAUIM. The average-utility of an itemset is the sum of the utilities of its items, in transaction where it appears, divided by its length (number of items). The average-utility model provides an alternative measure to assess the utility of itemsets. Because the average utility measure considers the length of itemsets, it is more suitable and applicable in real-life situations, than the traditional measure. Lin et al. [11] then developed a high average-utility pattern (HAUP)-tree structure to mine HAUIs more efficiently. Lan et al. [13] developed a projection-based average-utility (PBAU) mining algorithm to mine HAUIs. Lan et al. [14] then also extended the PBAU algorithm and designed a PAI approach using an improved strategy for mining the HAUIs. Lu et al. [16] then developed a HAU-tree structure to mine HAUIs without candidate generation.

The above algorithms were designed to mine HAUIs using a single minimum average-utility threshold (count). But in real-life situations, items are often regarded as having different importance to the user. Hence, it is unfair to assess the utility of all items using a same minimum average utility threshold (count), to determine if itemsets are HAUIs. In the past, the MSApriori [9] was first proposed to mine FIs under multiple minimum support thresholds. The CFP-growth algorithm [8] was then designed to build the MIS-tree and perform a recursive depth-first search to output the FIs. The MHU-Growth algorithm [17] extends CFP-Growth, to mine high utility frequent itemsets with multiple minimum support thresholds. Lin et al. [12] then developed the HUIM-MMU model for discovering HUIs with multiple minimum utility thresholds. Besides, two improved TID-index and EUCP strategies were proposed to prune unpromising itemsets early, thus speeding up the discovery of HUIs.

3 Preliminaries and Problem Statement

Let $I = \{i_1, i_2, \dots, i_r\}$ be a finite set of r distinct items occurring in a database D , and $D = \{T_1, T_2, \dots, T_n\}$ be a set of transactions, where for each transaction $T_q \in D$, T_q is a subset of I and has a unique identifier q , called its *TID* (transaction identifier). For each item i_j and transaction T_q , a positive number $q(i_j, T_q)$ represents the purchase quantity of i_j in transaction T_q . Moreover, a profit table $ptable = \{p(i_1), p(i_2), \dots, p(i_r)\}$ is defined, where $p(i_m)$ is a positive integer representing the unit profit of item i_m ($1 \leq m \leq r$). A set of k distinct items $X = \{i_1, i_2, \dots, i_k\}$ such that $X \subseteq I$ is said to be a k -itemset, where k is the length or level of the itemset. An itemset X is said to be contained in a transaction T_q if $X \subseteq T_q$. An example quantitative database is shown in Table 1. It consists of five transactions and six items, represented using letters from (a) to (f). The profit of each item in Table 2.

Table 1. A quantitative database.

TID	Items
1	$b:7, c:2, d:3, e:1$
2	$b:4, c:3, d:3$
3	$a:2, d:1$
4	$a:1, c:6, f:4$
5	$b:2, c:3, d:1, f:2$

Table 2. A profit table.

Item	Profit
a	5
b	2
c	1
d	2
e	4
f	1

Definition 1. The minimum average-utility threshold to be used for an item i_j in a database D is a positive integer denoted as $mau(i_j)$. A *MMAU-table* is used to store the minimum average-utility thresholds of all items in D , and is defined as:

$$MMAU - table = \{mau(i_1), mau(i_2), \dots, mau(i_r)\}, \quad (1)$$

In the following, it will be assumed that the minimum average-utility thresholds of all items for the running example are defined as: $\{mau(a):8, mau(b):8, mau(c):13, mau(d):14, mau(e):20, mau(f):9\}$

Definition 2. For a k -itemset X , the minimum average-utility threshold of X is denoted as $mau(X)$, and is defined as:

$$mau(X) = \frac{\sum_{i_j \in X} mau(i_j)}{|X|} = \frac{\sum_{i_j \in X} mau(i_j)}{k}. \quad (2)$$

Definition 3. The average-utility of an item i_j in a transaction T_q is denoted as $au(i_j, T_q)$, and is defined as:

$$au(i_j, T_q) = \frac{q(i_j, T_q) \times p(i_j)}{1}, \quad (3)$$

where $q(i_j, T_q)$ is the purchase quantity of item i_j in T_q , and $p(i_j)$ is the unit profit of item i_j .

Definition 4. The average-utility of a k -itemset X in a transaction T_q is denoted as $au(X, T_q)$, and defined as:

$$au(X, T_q) = \frac{\sum_{i_j \in X \wedge X \subseteq T_q} q(i_j, T_q) \times p(i_j)}{|X|} = \frac{\sum_{i_j \in X \wedge X \subseteq T_q} q(i_j, T_q) \times p(i_j)}{k}, \quad (4)$$

where k is the number of items in X .

Definition 5. The average-utility of an itemset X in a database D is denoted as $au(X)$, and is defined as:

$$au(X) = \sum_{X \subseteq T_q \wedge T_q \in D} au(X, T_q). \quad (5)$$

Problem Statement: The purpose of HAUI-MMAU is to efficiently discover the set of all high average-utility itemsets, where an itemset X is said to be a HAUI if its average utility is no less than its minimum average-utility threshold $mau(X)$ as:

$$HAUI \leftarrow \{X | au(X) \geq mau(X)\}. \quad (6)$$

4 Proposed HAUI-MMAU Framework

Based on the proposed HAUI-MMAU framework, a baseline algorithm for mining high-utility itemsets with multiple minimum average-utility thresholds (HAUI-MMAU) is proposed in this section. Thereafter, two improved algorithms based on two novel pruning strategies named IEUCP and PBCS are introduced to further increase mining efficiency.

4.1 Proposed Downward Closure Property

In HAUI-MMAU, the downward closure (DC) property does not hold for the average-utility measure. But a DC property can be restored using the average-utility upper bound ($auub$) model [7]. This allows to prune the search space early for discovering the HAUIs.

Definition 6. The average-utility upper bound of an itemset X is defined as the sum of the maximum utilities of transactions where X appears:

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

where $mu(T_q)$ is the maximum utility of transaction T_q , defined as $mu(T_q) = \max(q(i_j, T_q) \times p(i_j)) \forall i_j \in I$.

Definition 7. An itemset X is a high average-utility upper-bound itemset (HAUUBI) if its $auub$ is no less than its minimum average-utility threshold. The set of all HAUUBIs is thus defined as:

$$HAUUBI \leftarrow \{X | auub(X) \geq mau(X)\}. \quad (8)$$

Property 1. Let X^k and X^{k-1} be two itemsets respectively containing k and $k - 1$ items, such that $X^{k-1} \subset X^k$. The $auub$ property of HAUI-MMAU states that $auub(X^k) \leq auub(X^{k-1})$. Thus, if an itemset X^{k-1} is not a HAUUBI, then any superset X^k of X^{k-1} is also not a HAUUBI, nor a HAUI.

In HAUM, the *auub* property was shown to be very effective at reducing the number itemsets to be considered in the search space. However, for the problem of high average-utility itemset mining with multiple minimum average-utility thresholds (HAUM-MMAU), the *auub* property does not hold. The reason is that itemsets in different levels (itemsets of different lengths) may be evaluated using different minimum average-utility thresholds. Thus, the *auub* property of the HAUM framework cannot be directly adopted into the designed HAUM-MMAU framework. To address this limitation, this paper proposes the concept of least minimum average utility (*LMAU*), which is defined as follows.

Strategy 1. The items in the *MMAU-table* are sorted in ascending order of their *mau* values.

For example, items in the *MMAU-table* of the running example are sorted as $\{mau(a):8, mau(b):8, mau(f):9, mau(c):13, mau(d):14, mau(e):20\}$, that is by descending order of their *mau* values.

Based on the designed Strategy 1, the following transaction-maximum-utility downward closure (TMUDC) property holds for the HAUM-MMAU framework.

Theorem 1 (Transaction- Maximum-Utility Downward Closure Property, TMUDC Property). *Without loss of generality, assume that items in itemsets are sorted in ascending order of mau values. Let X^k be a k -itemset ($k \geq 2$), and X^{k-1} be a subset of X^k of length $k - 1$. If X^k is a HAUUBI, then X^{k-1} is also a HAUUBI.*

Although the TMUDC property can guarantee the anti-monotonicity for HAUUBIs, some HAUIs may still be missed. This can happen when a HAUUBI X of length 1 is evaluated using its *mau*(X) value. For example, assume that the *auub* value of an itemset (i_j) is less than *mau*(i_j). This itemset will thus be considered as not being an HAUUBI. Moreover, according to the TMUDC property, any extension Y of the itemset (i_j) is also not a HAUUBI. Consider that $Y = \{i_j, i_z\}$ such that the item i_z has a lower *mau* value than i_j . Then Y would still be a HAUI if $au(Y) \geq \frac{mau(i_j)+mau(i_z)}{2}$. Thus, if the TMUDC is directly applied to obtain the HAUUBIs of length 1, Y would not be considered as a HAUUBI since *auub*(i_j) < *mau*(i_j). As a result, Y would not be included into the final set of HAUIs. Hence, it is incorrect to apply the TMUDC property to determine the HAUUBIs of length 1. To solve this problem, the concept of *LMAU* value is introduced, which guarantee the discovery of the complete set of HAUIs in the designed HAUM-MMAU framework.

Definition 8 (Least Minimum Average-Utility, LMAU). The least minimum average-utility (LMAU) in the *MMAU-table* is defined as:

$$LMAU = \min\{mau(i_1), mau(i_2), \dots, mau(i_r)\}. \quad (9)$$

where r is the total number of items.

The *LMAU* in the running example is calculated as $LMAU = \min\{mau(a), mau(b), mau(f), mau(c), mau(d), mau(e)\} = \min\{8, 8, 9, 13, 14, 20\} (= 8)$.

Theorem 2 (HAUIs \subseteq HAUUBIs). *Let be an itemset X^{k-1} of length $k - 1$ and X^k be one of its supersets. If X^{k-1} has an *auub* value lower than the *LMAU*, X^{k-1} is not a HAUUBI nor a HAUI, as well as all its supersets. Hence, X^{k-1} and its supersets can be discarded.*

The above theorem indicates that any superset of a non-HAUUBI cannot be a HAUUBI. That is, only the combination of two HAUUBIs may generate a potential HAUI. By using this approach for candidate generation, many unpromising candidates can be eliminated, and it becomes unnecessary to calculate their actual average-utility value in the later mining process.

4.2 The Proposed HAUI-MMAU Algorithm

The designed baseline HAUI-MMAU algorithm consists of two phases. In the first phase, the designed HAUI-MMAU algorithm performs a breadth-first search to mine the HAUUBIs. In the second phase, an additional database scan is performed to identify the actual high average-utility itemsets from the set of HAUUBIs discovered in the first phase. The pseudocode of the proposed algorithm is given in Algorithm 1.

Algorithm 1. HAUI-MMAU

Input: D , a quantitative transactional databases; $ptable$, a profit table;
 $MMAU-table$, the user predefined multiple minimum average utility
threshold table.

Output: The set of complete high average-utility itemsets (HAUIs).

```

1 find the LMAU in the MMAU-table;
2 scan  $D$  to find  $auub(i_j)$ ;
3 for each item  $i_j$  do
4   if  $auub(i_j) \geq LMAU$  then
5      $HAUUBI^1 \leftarrow HAUUBI^1 \cup i_j$ ;
6 sort items in  $HAUUBI^1$  in ascending order of their mau values;
7 set  $k \leftarrow 2$ ;
8 while  $HAUUBI^{k-1} \neq null$  do
9    $C_k = generate\_candidate(HAUUBI^{k-1})$ ;
10  for each  $k$ -itemset  $X \in C_k$  do
11    scan  $D$  to calculate  $auub(X)$ ;
12    if  $auub(X) \geq mau(X)$  then
13       $HAUUBI^k \leftarrow HAUUBI^k \cup X$ ;
14  set  $k \leftarrow k + 1$ ;
15  $HAUUBIs \leftarrow \bigcup HAUUBI^k$ ;
16 for each itemset  $X$  in  $HAUUBIs$  do
17   scan  $D$  to calculate  $au(X)$ ;
18   if  $au(X) \geq mau(X)$  then
19      $HAUIs \leftarrow HAUIs \cup X$ ;
20 return  $HAUIs$ ;
```

The proposed HAU-MMAU algorithm takes as input: (1) a quantitative transactional database D (2) a profit table $ptable$ indicating the unit profit of each item, and (3) a multiple minimum average-utility threshold table, $MMAU-table$. First, the least minimum average-utility value ($LMAU$) is found in the $MMAU-table$ (Line 1). Then, the database is scanned to find the $auub$ values of all 1-items (Line 2). For each 1-item, the algorithm check if its $auub$ value is no less than the $LMAU$, to determine if it is an $HAUUBI$ of length 1 (Lines 3 to 5). After obtaining the set of all $HAUUBIs$ of length 1 ($HAUUBI^1$), the items in $HAUUBI^1$ are sorted in ascending order of their mau values (Line 6). This process is necessary to ensure that all $HAUIs$ will be retrieved, as explained in the previous section. The parameter k is then set to 2 (Line 7), and a loop is performed to mine all $HAUUBIs$ in a level-wise way, starting from itemsets of length $k = 2$ (Lines 8 to 14). During the $(k-1)$ -th iteration of the loop, the set $HAUUBI^k$, containing the $HAUUBIs$ of length k , is obtained by the following process. First, pairs of $(k-1)$ -itemsets in $HAUUBI^{k-1}$ are combined to generate their supersets of length k by applying the *generate_candidate* procedure (Line 9). The result is a set named C_k , containing potential $HAUUBIs$ of length k . The database is then scanned again to calculate the $auub$ value of each itemset X in C_k . If the itemset X has an $auub$ value that is no less than its maximum average-utility (mau), it is added to the set $HAUUBI^k$ (Lines 11 to 13). This loop terminates when no more candidates can be generated.

In the second phase, the database is scanned again to find the high average-utility itemsets ($HAUIs$) in the set of $HAUUBIs$, found in first phase (Lines 16 to 19). Based on the proposed theorems, the designed algorithm is correct and complete, and thus returns the full set of $HAUIs$ (Line 20).

4.3 Improved IEUCP Strategy

The TMUDC property proposed in the designed HAU-MMAU algorithm can considerably reduce the search space and speed up the discovery of $HAUIs$. However, the designed HAU-MMAU algorithm still suffers from the combinational explosion of the number of itemsets in the search space, since it generates candidates in a level-wise way, and many candidates may be generated at each level. To address this limitation, the Improved estimated utility co-occurrence pruning strategy (IEUCP) is designed to reduce the number of join operations (the operations of combining pairs of itemsets to generate larger itemsets) by pruning unpromising itemsets early. The IEUCP strategy is applied for the generation of k -itemsets ($k \geq 2$). It relies on the following theorem to ensure the correctness and the completeness of the HAU-MMAU algorithm with the IEUCP strategy, for discovering $HAUIs$.

Theorem 3 (Improved Estimated Utility Co-occurrence Pruning Strategy, IEUCP). *Let there be an itemset X^{k-1} and an itemset X^k that is an extension of X^{k-1} (that was obtained by appending an item to X^{k-1}). Without loss of generality, assume that items in itemsets are sorted by ascending order of mau values. A sorted prefix of X^{k-m} is a subset of X^{k-1} containing*

the m first items of X^{k-1} according to ascending order of mau values, where $1 \leq m < k - 1$. For example, consider the itemset $(bcde)$, which is a 4-itemset ($k = 4$). Its sorted prefixes are (b) , (bc) , and (bcd) . If there exists a sorted prefix X^{k-m} of X^k that is not a HAUUBI, then X^k is also not a HAUUBI.

It is important to note that an itemset X can only be safely pruned by Theorem 3 if X has a sorted prefix that is not a HAUUBI, and that this pruning condition should not be applied by considering subsets of X that are not sorted prefixes of X . Based on Theorem 3 and the proposed TMUDC property, the completeness and correctness of the HAU-MMAU algorithm is preserved, when applying the IEUCP strategy.

4.4 Pruning Before Calculation Strategy, PBCS

Although the developed IEUCP strategy is effective at reducing the number of unpromising candidates in the first phase, calculating the set of actual HAUIs in the second phase remains a very time-consuming process. To address this issue, this article further proposes an efficient pruning before calculation strategy (PBCS) to prune itemsets without performing a database scan.

Theorem 4. *Let X , X^a , and X^b be itemsets such that $X^a \subset X$, $X^b \subset X$, and $X^a \cup X^b = X$, $X^a \cap X^b = \emptyset$. If both X^a and X^b are not HAUIs, the itemset X is also not a HAUI.*

Thus, if it is found that two subsets X^a and X^b of an itemset X are not HAUIs and respect the condition of Theorem 4, it can be concluded that X is not a HAUI. It is thus unnecessary to scan the database to calculate the actual utility of X . Based on this PBCS, the cost of calculating the actual utility of HAUUBIs in the second phase can be greatly reduced.

5 Experimental Evaluation

Substantial experiments were conducted to evaluate the effectiveness and efficiency of the proposed algorithms. The performance of the baseline HAU-MMAU algorithm was compared with two versions named HAU-MMAU_{IEUCP} and HAU-MMAU_{PBCS}, which respectively integrates the proposed IEUCP strategy, and both the designed IEUCP and PBCS strategies. Note that no previous studies have been done for mining HAUIs with multiple average-utility thresholds, and it is thus unnecessary to compare the designed algorithms with the previous algorithms for HAUIM with a single minimum average-utility threshold. Experiments were carried on both real-life and synthetic dataset, having various characteristics. The foodmart, retail, and chess datasets were obtained from the SPMF website [6]. The IBM Quest Synthetic Dataset Generator [2] was used to generate a synthetic dataset named T40I10D100K. Parameters and characteristics of the datasets used in the experiments are respectively shown in Tables 3 and 4.

Table 3. Parameters of the datasets.

$\# D $	Transaction count
$\# I $	Number of distinct items
AvgLen	Average transaction length
MaxLen	Maximal transaction length
Type	Dataset type

Table 4. Characteristics of the datasets.

Dataset	$\# D $	$\# I $	AvgLen	MaxLen	Type
Foodmart	21,556	1,559	4.4	10	Sparse
Retail	88,162	16,470	10.3	76	Sparse
Chess	3,196	75	36	36	Dense
T40I10D100K	100,000	942	39.6	77	Dense

Furthermore, the method proposed in [9] for assigning the multiple thresholds to items was adapted to automatically set the *mau* value of each item in the proposed HAUI-MMAU framework. The following equation is thus used to set the *mau* value of each item i_j :

$$mau(i_j) = \max\{\beta \times p(i_j), GLMAU\}, \quad (10)$$

where β is a constant used to set the *mau* values of an item as a function of its unit profit. To ensure randomness and diversity in the experiments, β was respectively set in different interval for varied datasets. The constant *GLMAU* is user-specified and represents the global least average-utility value. Lastly, $p(i_j)$ represents the external utility (unit profit) of the item i_j . If β is set to zero, a single minimum average-utility threshold *GLMAU* is used for all items. In that case, the task of HAUI-MMAU would become the same as traditional HAUI.

5.1 Runtime

In this section, the runtimes of three proposed approaches are respectively compared. Figure 1 shows the runtime of the proposed algorithms when β is varied with a fixed *GLMAU* value within the predefined interval for each dataset.

It can be observed in Fig. 1 that the improved HAUI-MMAU_{IEUCP} and HAUI-MMAU_{PBCS} algorithms outperform the baseline HAUI-MMAU algorithm. For example, when *GLMAU* is set to 168,000 and β is set in the [200, 400] interval for the chess dataset, the runtime of HAUI-MMAU, HAUI-MMAU_{IEUCP} and HAUI-MMAU_{PBCS} are respectively 116, 133 and 58 seconds. It can also be observed that HAUI-MMAU_{PBCS} always has the best performance among the three algorithms. The reason is that HAUI-MMAU_{PBCS} applies both the IEUCP and PBCS strategies to improve the performance of the mining process,

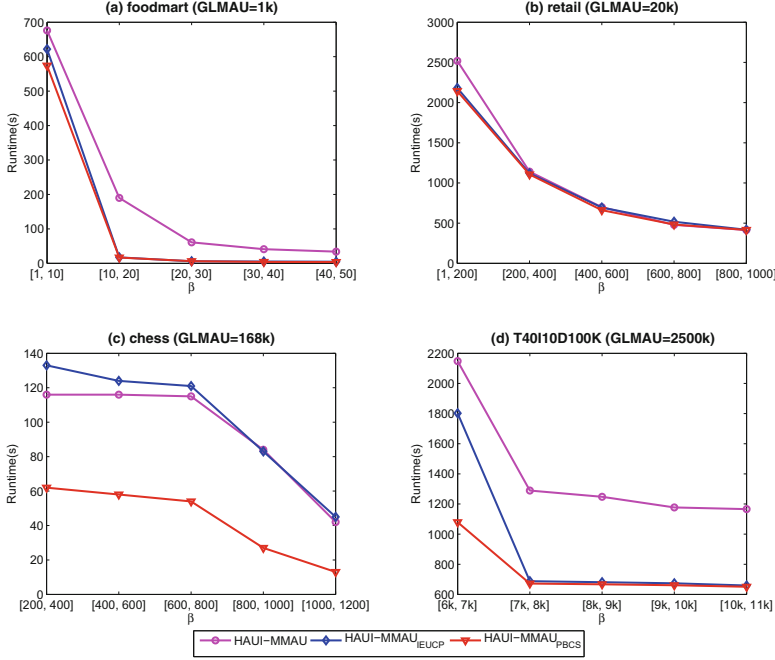


Fig. 1. Runtime for a fixed $GLMAU$ and varied β .

while HAU-MMAU_{IEUCP} only applies the IEUCP strategy. But both strategies are complementary to each other. The IEUCP strategy can prune a large amount of unpromising itemsets to avoid several database scans when calculating the $auub$ values of itemsets in the first phase, while the PBCS strategy can prune unpromising HAUIs from the remaining $HAUUBIs$ to avoid calculating their actual average-utility values, in the second phase. Hence, HAU-MMAU_{PBCS} has the best performance among the three algorithms. In Fig. 1, it can also be observed that the performance gap between HUAUI-MMAU_{PBCS} and the other algorithms is larger for the chess and T40I10D100K datasets, especially when β is set lower. The reason is that both chess and T40I10D100K are dense datasets with relatively few distinct items. As a result, the discovered HAUUBIs are highly related to each other and the designed PBCS strategy can thus be used to prune the unpromising HAUIs early from the remaining HAUUBIs.

5.2 Candidate Analysis

We further compared the number of candidates generated by each of the three algorithms. Here, an itemset is said to be a candidate if its actual average-utility value is calculated in the first phase, and its exact average-utility is calculated in the second phase. Figure 2 shows the number of candidates for the three algorithms when β is varied with a fixed $GLMAU$ value.

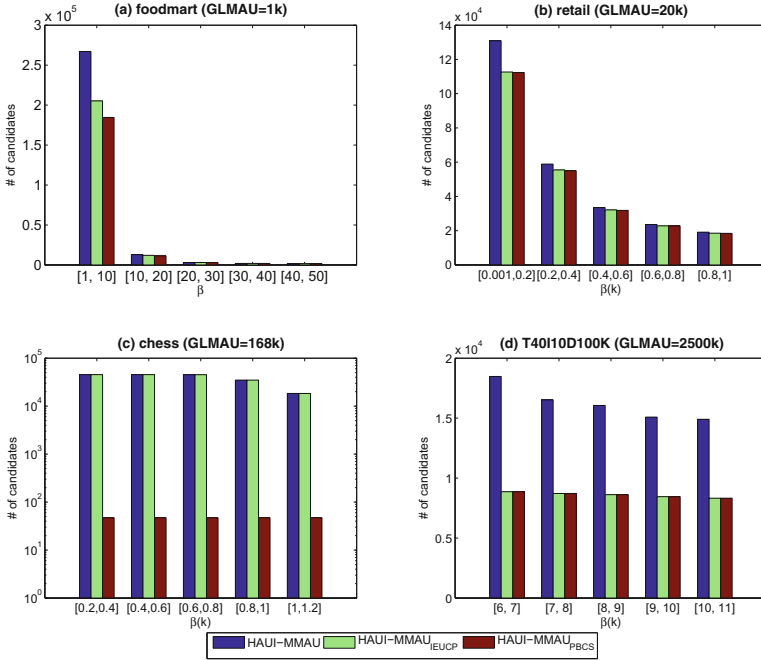


Fig. 2. Number of candidates for a fixed $GLMAU$ and varied β

In Fig. 2, it can be observed that the HAU-MMAU algorithm always considers more candidates than the other two algorithms. The reason is that HAU-MMAU_{IEUCP} can efficiently prune some unpromising HAUUBIs using its designed pruning strategy, while HAU-MMAU_{PBCS} can prune unpromising HAUIs from the remaining HAUUBIs for revealing the actual HAUIs. It can also be found in Fig. 2(a) that when β is set in the $[1, 10]$ interval, more candidates are generated than for the other intervals of β such as $[10, 20]$, $[20, 30]$, $[30, 40]$ and $[40, 50]$. The reason is that foodmart is a very sparse dataset having a large number of distinct items. In the $[1, 10]$ interval, the generated candidates have similar average-utility values. Since the minimum average-utility threshold is set lower, the amounts of candidates are thus revealed in this interval. When the minimum average-utility threshold is increased, itemsets with similar average-utility values may be lower than the certain threshold; a large amount of candidates can thus be pruned. It can be observed in Fig. 2(b) and (d), that the number of candidates considered by HAU-MMAU_{IEUCP} and HAU-MMAU_{PBCS} is almost the same. This explains why they have similar runtimes, as shown in Fig. 1(b) and (d). But in Fig. 2(c), it is obvious that the number candidates considered by HAU-MMAU_{PBCS} is much less than the two other algorithms, which shows that the proposed PBCS strategy can greatly reduce the search space in the second phase for the chess dataset.

5.3 Memory Usage

This section compares the memory usage of the three designed algorithms when β is varied and $GLMAU$ is fixed. Results are shown in Fig. 3.

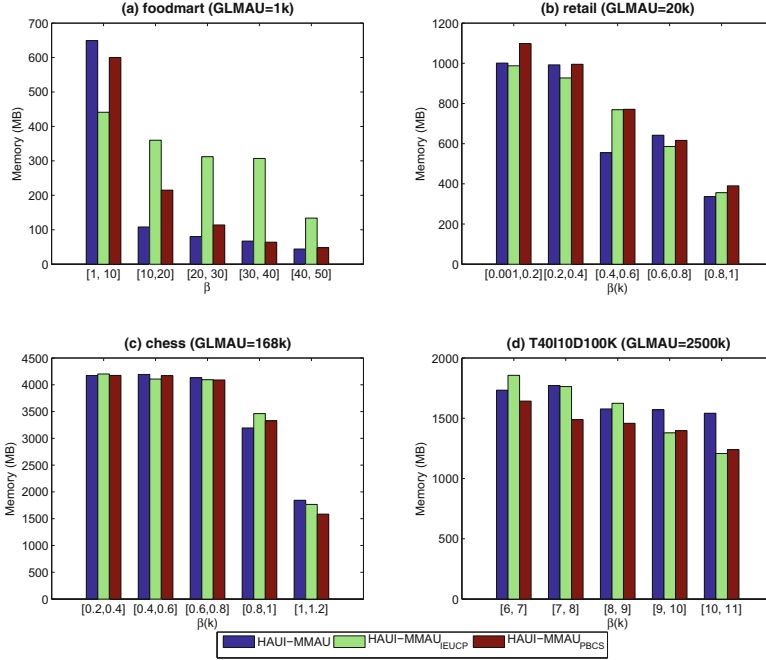


Fig. 3. Memory usage for a fixed $GLMAU$ and various values of β .

It can be seen in Fig. 3 that HAU-MMAU sometimes consumes less memory than HAU-MMAU_{IEUCP} when β is varied and $GLMAU$ is fixed. In most cases, the HAU-MMAU_{PBCS} algorithm consumes more memory than the other two algorithms since both the IEUCP and PBCS strategies are adopted to prune unpromising candidates, as it can be observed in Fig. 3(b) and (c). Although the IEUCP strategy requires additional memory to store its structure, pruning itemsets reduces the number of itemsets to then be considered as potential HAUIs, and thus the runtime and memory usage. Thus, the HAU-MMAU_{PBCS} algorithm consumes less memory than the HAU-MMAU and HAU-MMAU_{IEUCP} algorithms, in most cases.

6 Conclusion

In this paper, the high average-utility itemset mining with multiple minimum average-utility thresholds (HAUIM-MMAU) framework was designed to mine

high average-utility itemsets (HAUIs) with multiple minimum average-utility thresholds. The baseline HAUI-MMAU algorithm is a two phases algorithm, which relies on several designed theorems to find the HAUIs. The first IEUCP strategy is designed to prune the search space, and thus increasing the efficiency of HAUI mining. The second PBCS pruning strategy is used to reduce the number of HAUIs at the beginning of the second phase, for revealing the actual HAUIs. An extensive experimental study was conducted on both synthetic and real datasets to evaluate the performance of the algorithms in terms of runtime, number of candidates, and memory usage. Results show that the designed algorithms can efficiently discover the HAUIs and that the two pruning strategies can effectively reduce the number of candidates in the first and second phase.

References

1. Agarwal, R., Imielinski, T., Swami, A.: Database mining: a performance perspective. *IEEE Trans. Knowl. Data Eng.* **5**(6), 914–925 (1993)
2. Agrawal, R., Srikant, R.: Quest synthetic data generator (1994). <http://www.Almaden.ibm.com/cs/quest/syndata.html>
3. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: *International Conference on Very Large Data Bases*, pp. 487–499 (1994)
4. Chan, R., Yang, Q., Shen, Y.D.: Mining high utility itemsets. In: *IEEE International Conference on Data Mining*, pp. 19–26 (2003)
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: *Andreasen, T., Christiansen, H., Cubero, J.-C., Raś, Z.W. (eds.) ISMIS 2014. LNCS, vol. 8502*, pp. 83–92. Springer, Heidelberg (2014)
6. SPMF: an open-source data mining library. <http://www.philippe-fournier-viger.com/spmf/>
7. Hong, T.P., Lee, C.H., Wang, S.L.: Effective utility mining with the measure of average utility. *Expert Syst. Appl.* **38**(7), 8259–8265 (2011)
8. Kiran, R.U., Reddy, P.K.: Novel techniques to reduce search space in multiple minimum supports-based frequent pattern mining algorithms. In: *ACM International Conference on Extending Database Technology*, pp. 11–20 (2011)
9. Liu, B., Hsu, W., Ma, Y.: Mining association rules with multiple minimum supports. In: *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 337–341 (1999)
10. Liu, Y., Liao, W., Choudhary, A.K.: A two-phase algorithm for fast discovery of high utility itemsets. In: *Ho, T.-B., Cheung, D., Liu, H. (eds.) PAKDD 2005. LNCS (LNAI), vol. 3518*, pp. 689–695. Springer, Heidelberg (2005)
11. Lin, C.-W., Hong, T.-P., Lu, W.-H.: Efficiently mining high average utility itemsets with a tree structure. In: *Nguyen, N.T., Le, M.T., Świątek, J. (eds.) ACIIDS 2010. LNCS, vol. 5990*, pp. 131–139. Springer, Heidelberg (2010)
12. Lin, J.C.W., Gan, W., Fournier-Viger, P., Hong, T.P.: Mining high-utility itemsets with multiple minimum utility thresholds. In: *International C* Conference on Computer Science and Software Engineering*, pp. 9–17 (2015)
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**(1), 193–209 (2012)

14. Lan, G.C., Hong, T.P., Tseng, V.S.: Efficiently mining high average-utility itemsets with an improved upper-bound strategy. *Int. J. Inf. Technol. Decis. Making* **11**(5), 1009–1030 (2012)
15. Liu, M., Qu, J.: Mining high utility itemsets without candidate generation. In: *ACM International Conference on Information and Knowledge Management*, pp. 55–64 (2012)
16. Lu, T., Vo, B., Nguyen, H.T., Hong, T.-P.: A new method for mining high average utility itemsets. In: Saeed, K., Snášel, V. (eds.) *CISIM 2014. LNCS*, vol. 8838, pp. 33–42. Springer, Heidelberg (2014)
17. Ryang, H., Yun, U., Ryu, K.: Discovering high utility itemsets with multiple minimum supports. *Intell. Data Anal.* **18**(6), 1027–1047 (2014)
18. Yao, H., Hamilton, H.J., Butz, C.J.: A foundational approach to mining itemset utilities from databases. In: *SIAM International Conference on Data Mining*, pp. 211–225 (2004)
19. Yao, H., Hamilton, H.J.: Mining itemset utilities from transaction databases. *Data Knowl. Eng.* **59**(3), 603–626 (2006)