

# More Efficient Algorithms for Mining High-Utility Itemsets with Multiple Minimum Utility Thresholds

Wensheng Gan<sup>1</sup>, Jerry Chun-Wei Lin<sup>1(✉)</sup>, Philippe Fournier-Viger<sup>2</sup>,  
and Han-Chieh Chao<sup>1,3</sup>

<sup>1</sup> School of Computer Science and Technology,  
Harbin Institute of Technology Shenzhen Graduate School, Shenzhen, China  
wsgan001@gmail.com, jerrylin@ieee.org, hcc@ndhu.edu.tw

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

<sup>3</sup> Department of Computer Science and Information Engineering,  
National Dong Hwa University, Hualien, Taiwan

**Abstract.** Mining high-utility itemsets (HUIs) is a popular data mining task, which consists of discovering sets of items that yield a high profit in a transaction database. Although HUI mining has numerous applications, a key limitation is that a single minimum utility threshold (*minutil*) is used to assess the utility of all items. This simplifying assumption is unrealistic since in real-life all items do not have the same unit profit, and thus do not have an equal chance of generating a high profit. As a result, if the *minutil* threshold is set high, patterns containing items having a low unit profit are often missed, while if *minutil* is set low, the number of patterns becomes unmanageable. To address this issue, this paper presents an efficient tree-based algorithm named HIMU for mining HUIs using multiple minimum utility thresholds. A novel tree structure called multiple item utility Set-enumeration (MIU)-tree and the *global and conditional downward closure (GDC and CDC) properties* of HUIs in the MIU-tree are proposed. Moreover, a vertical compact utility-list structure is adopted to store the information required for discovering HUIs without performing additional database scans and generating candidates. An extensive experimental study on real-world and synthetic datasets show that this greatly improves the efficiency of the algorithm in terms of runtime and scalability.

**Keywords:** High-utility itemsets · Rare item problem · Set-enumeration tree · *SDC property* · Pruning strategies

## 1 Introduction

Knowledge Discovery in Database (KDD) is the process of finding meaningful, unexpected, and useful information in large amounts of data [2, 3]. Two fundamental tasks in KDD are frequent itemset mining (FIM) and

association rule mining (ARM) [2, 3], which have numerous applications, in many domains. In contrast with traditional FIM and ARM, high-utility itemset mining (HUIM) [5, 6, 10, 11, 13, 14, 16] considers that items may have different unit profits and that purchase quantities may be non binary, to measure how “useful” an item or itemset is. The “utility” of an itemset in HUIM represents its importance to users in real-life applications (e.g., weight, cost, risk, or unit profit). The goal of HUIM is to identify itemsets in transactions that may be frequent or rare, but yield a high profit. HUIM is a key data analysis task, and has been widely utilized to discover valuable knowledge in several domains. Many approaches have been developed to mine high-utility itemsets such as Two-Phase [11], IHUP [5], UP-growth [13], UP-growth+ [14], HUI-Miner [10], and FHM [8], and so on.

However, an important limitation of previous studies is that they rely on a single minimum utility threshold to discover the complete set of HUIs. Using a single threshold value to assess the utility of all items in a database is inadequate since each item is different and thus items should not all be treated the same. Traditional high-utility itemset mining algorithms only let the user specify one *minutil* threshold to assess the utility of all patterns. Using a single threshold implies that all patterns in the database should have an equal chance of having a utility higher than the *minutil* threshold. But this assumption is unrealistic in practical applications [9] since each item generally has a distinct nature, frequency, or importance, and thus different items may tend to exhibit a lower or a higher utility. Hence, using a single fixed threshold, it is difficult to fairly measure the utility of items or itemsets. For example, in a retail store, it may be desirable to view the itemset  $\{diamond\}$  as a HUI if it brings more than 5,000\$/week, but to view the itemset  $\{bread, milk\}$  as a HUI if its profit is greater than 100\$/week. Using traditional HUI mining algorithms, if the *minutil* threshold is set high, useful patterns having a low utility are missed, and if it is set low, the number of HUIs becomes unmanageable. Thus, assessing the utility of items using a single threshold is inadequate as it does not take the inherent nature of each item (i.e., utility, item importance) into account. It is a non-trivial task and an important challenge to design efficient algorithms that solve this issue.

Mining association rules and frequent itemsets using multiple minimum support thresholds has been extensively studied [7, 12, 15], but the proposed approaches cannot be directly used in HUIM since HUIM considers non binary purchase quantities, and the unit profits of items. Up to now, few works have addressed the problem of mining HUIs with multiple minimum utility thresholds. To the best of our knowledge, HUI-MMU and the improved HUI-MMU<sub>TID</sub> algorithms [9] are the only algorithms designed to address this issue. However, a drawback of these algorithms is that they use a level-wise candidate generation-and-test approach to mine HUIs, which may perform poorly on databases containing long transactions or when minimum utility thresholds are set low. In this paper, to improve the efficiency of HUIM with multiple thresholds, an efficient tree-based algorithm named mining **H**igh-utility **I**temsets with **M**ultiple minimum **U**tility thresholds (abbreviated as **HIMU**) is developed. The contributions of this work are fourfold:

- A fast algorithm named HIMU is proposed to reveal useful and meaningful High-utility Itemsets by considering Multiple minimum Utility thresholds. The user can assign a minimum utility threshold to each item based on its real-life utility. This is more flexible and realistic than using a single *minutil* threshold.
- In contrast with previous Apriori-based algorithms, the proposed HIMU algorithm avoids repeatedly scanning the database and generating candidates, thanks to a novel sorted Set-enumeration tree structure named MIU-tree, and the use of a compact utility-list structure that allows obtaining information about an itemset by combining utility-lists of its prefix itemsets.
- Moreover, two novel *global and conditional sorted downward closure (GDC and CDC) properties* guarantee the global and partial anti-monotonicity for mining HUIs in the MIU-tree. Thus, HIMU can easily discover HUIs while pruning a huge number of unpromising itemsets, and only two database scans are performed by HIMU, which is more efficient than previous algorithms.
- Extensive experiments on two real-world datasets show that the proposed algorithms efficiently discover HUIs and outperform the state-of-the-art HUI-MMU and HUI-MMU<sub>TID</sub> algorithms. In addition, the improved algorithm outperforms the baseline algorithm, in terms of runtime and scalability.

## 2 Related Works

High-utility itemset mining (HUIM) considers the internal transaction utilities (purchase quantities) and external utilities (unit profits) of items to discover the profitable itemsets in quantitative databases. HUIM was introduced by Chan et al. [6]. Yao et al. then defined a strict unified framework for HUIM [16]. Since the *downward closure property* of ARM does not hold in HUIM, Liu et al. designed the TWU model [11] and a transaction-weighted downward closure (TWDC) property, to greatly reduce the number of unpromising candidates when mining HUIs using a level-wise approach. Several tree-based approaches for HUIM such as IHUP [5], UP-growth [13] and UP-growth+ [14] have been proposed. These pattern-growth approaches, however, generate and keep a huge number of candidates in memory to then obtain the actual HUIs. To address the above limitations of traditional HUIM, the HUI-Miner algorithm was proposed to directly mine HUIs while avoiding performing multiple database scans and generating candidates based on a designed utility-list structure [10]. The FHM algorithm was further proposed to enhance the performance of HUI-Miner using co-occurrences of pair of items [8].

Besides traditional HUIM, several variations of HUIM have been developed. The development of algorithms for HUIM is an active research topic, but most of them consider a single *minutil* threshold. In the field of FIM, several algorithms have been designed to address the “rare item problem such as MSApriori [7], CFP-growth [15], and CFP-growth++ [12]. The key idea of these works is to extract frequent patterns involving rare items using the “multiple minimum supports framework” [7, 12, 15]. This framework allows the user to specify multiple minimum support thresholds to take into account the nature of each item in

**Table 1.** An example database

TID	Transaction
$T_1$	$a:1, c:2, d:3$
$T_2$	$a:2, d:1, e:2$
$T_3$	$b:3, c:5$
$T_4$	$a:1, c:3, d:1, e:2$
$T_5$	$b:1, d:3, e:2$
$T_6$	$b:2, d:2$
$T_7$	$b:3, c:2, d:1, e:1$
$T_8$	$a:2, c:3$
$T_9$	$c:2, d:2, e:1$
$T_{10}$	$a:2, c:2, d:1$

**Table 2.** Derived HUIs

Itemset	MIU	Utility	Itemset	MIU	Utility
$(b)$	65	108	$(de)$	50	96
$(d)$	50	126	$(acd)$	50	76
$(ad)$	50	90	$(bde)$	50	93
$(bc)$	53	79	$(cde)$	50	55
$(bd)$	50	126	$(bcde)$	50	50
$(cd)$	50	83			

terms of frequency in the database. However, these approaches cannot be directly used in HUIM since HUIM requires to consider the purchase quantities and unit profits of items. There is only one paper that has considered the constraint of multiple minimum utility thresholds for mining HUIs [9].

### 3 Preliminaries and Problem Statement

Let  $I = \{i_1, i_2, \dots, i_m\}$  be a finite set of  $m$  distinct items appearing in a transactional database  $D = \{T_1, T_2, \dots, T_n\}$ , where each transaction  $T_q \in D$  is a subset of  $I$ , and has a unique identifier called its *TID*. A unit profit  $pr(i_j)$  is assigned to each item  $i_j \in I$ , which represents its importance (e.g. profit, interest, risk). Unit profits are stored in a profit-table  $ptable = \{pr(i_1), pr(i_2), \dots, pr(i_m)\}$ . An itemset  $X \subseteq I$  with  $k$  distinct items  $\{i_1, i_2, \dots, i_k\}$  is of length  $k$  and is referred to as a  $k$ -itemset. An itemset  $X$  is said to be contained in a transaction  $T_q$  if  $X \subseteq T_q$ . For an itemset  $X$ , let the notation  $TIDs(X)$  denotes the *TIDs* of transactions in  $D$  containing  $X$ . For example, Table 1 shows a transactional database containing 10 transactions, and will be used as running example. Assume that the profit-table is defined as in the  $ptable = \{pr(a) : 6, pr(b) : 12, pr(c) : 1, pr(d) : 9, pr(e) : 3\}$ .

**Definition 1.** The minimum utility threshold of an item  $i_j$  in a database  $D$  is denoted as  $mu(i_j)$ . A structure called *MMU-table* indicates the minimum utility thresholds of each item in  $D$ , and is defined as:

$$MMU\text{-table} = \{mu(i_1), mu(i_2), \dots, mu(i_m)\}. \quad (1)$$

Assume that the minimum utility thresholds of items in the running example are defined as:  $MMU\text{-table} = \{mu(a), mu(b), mu(c), mu(d), mu(e)\} = \{56, 65, 53, 50, 70\}$ . To avoid the ‘‘rare item problem’’, we consider the smallest utility threshold among items in an itemset as its minimum utility threshold, as defined below.

**Definition 2.** The minimum utility threshold of a  $k$ -itemset  $X = \{i_1, i_2, \dots, i_k\}$  in  $D$  is denoted as  $MIU(X)$ , and defined as the smallest  $mu$  value for items in  $X$ , that is:

$$MIU(X) = \min\{mu(i_j) | i_j \in X, 1 \leq j \leq k\}. \quad (2)$$

For example,  $MIU(a) = \min\{mu(a)\} = 56$ ,  $MIU(ac) = \min\{mu(a), mu(c)\} = \min\{56, 53\} = 53$ , and  $MIU(ace) = \min\{mu(a), mu(c), mu(e)\} = 53$ .

**Definition 3.** The utility of an item  $i_j$  in a transaction  $T_q$  is defined as:

$$u(i_j, T_q) = q(i_j, T_q) \times pr(i_j). \quad (3)$$

**Definition 4.** The utility of an itemset  $X$  in a transaction  $T_q$  is defined as:

$$u(X, T_q) = \sum_{i_j \in X \wedge X \subseteq T_q} u(i_j, T_q). \quad (4)$$

**Definition 5.** The utility of an itemset  $X$  in a database  $D$  is defined as:

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

**Definition 6.** The transaction utility of a transaction  $T_q$  is defined as:

$$tu(T_q) = \sum_{i_j \in T_q} u(i_j, T_q). \quad (6)$$

**Definition 7.** The transaction-weighted utility of an itemset  $X$  is denoted as  $TWU(X)$ , and defined as:

$$TWU(X) = \sum_{X \subseteq T_q \wedge T_q \in D} tu(T_q). \quad (7)$$

**Definition 8.** An itemset  $X \subseteq I$  is a high transaction-weighted utilization itemset (HTWUI) if its  $TWU$  value is no less than the minimum utility threshold [14]. To adapt this definition, we assume that this threshold is  $MIU(X)$ .

$$HTWUI \leftarrow \{X | TWU(X) \geq MIU(X)\}. \quad (8)$$

**Definition 9.** An itemset  $X$  in a database  $D$  is a high-utility itemset (HUI) if and only if its utility is no less than its minimum utility threshold:

$$HUI \leftarrow \{X | u(X) \geq MIU(X)\}. \quad (9)$$

For the running example, the complete set of HUIs when considering multiple minimum utility thresholds is shown in Table 2.

**Definition 10.** Given a transactional database  $D$  and a  $MMU$ -table, which defines the minimum utility thresholds of each item in  $D$ . The problem of mining high-utility itemsets in  $D$  with multiple minimum utility thresholds (HUI-MMU) is to find each itemset  $X$  having a utility no less than its threshold  $MIU(X)$ .

## 4 Proposed HIMU Algorithm for Mining HUIs

### 4.1 Search Space of HIMU and the Proposed MIU-Tree

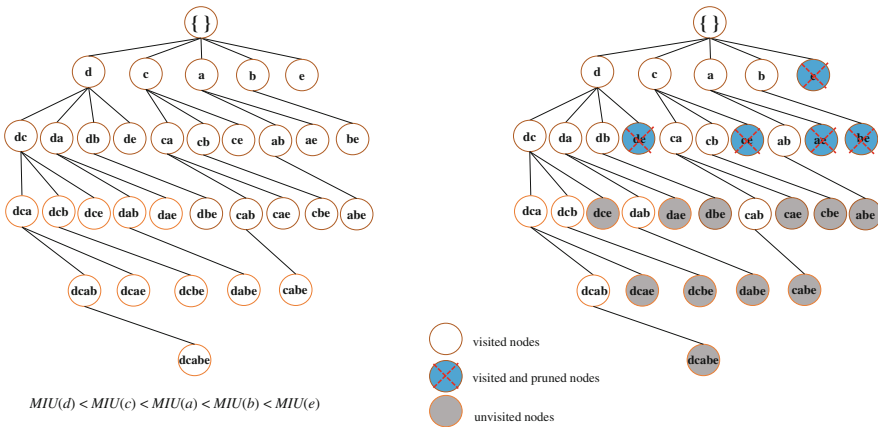
**Definition 11 (Total Order  $\prec$  on Items).** The proposed MIU-tree structure relies on a total order  $\prec$  on items. Assume that this order is the ascending order of minimum utility thresholds of items.

**Definition 12 (Set-Enumeration Tree with Multiple Minimum Item Utilities, MIU-Tree).** The designed MIU-tree structure is a sorted set-enumeration tree where the total order  $\prec$  on items is the ascending order of minimum utility thresholds of items.

**Definition 13.** The extensions (descendant nodes) of an itemset (tree node)  $X$  can be obtained by appending an item  $y$  to  $X$  such that  $y$  is greater than all items already in  $X$  according to the total order  $\prec$ .

For example, the proposed MIU-tree used by the HIMU algorithm for the running example is shown in Fig. 1 (left). Based on the designed MIU-tree, the following lemmas can be obtained.

**Lemma 1.** *The complete search space of the proposed HIMU algorithm for the HUIM-MMU framework can be represented by a MIU-tree where items are sorted according to the ascending order of the  $mu$  values on items.*



**Fig. 1.** The MIU-tree representation of the search space.

The traditional TWDC property of the TWU model does not hold in the proposed HUIM-MMU framework. For example, consider items  $(b)$ ,  $(c)$ ,  $(d)$  and  $(e)$  ( $MIU(b) : 65$ ,  $MIU(c) : 53$ ,  $MIU(d) : 50$  and  $MIU(e) : 70$ ). The TWU of an itemset  $(bce)$  is calculated as  $TWU(bce) = 50$ , which is less than the minimum utility values of its subsets  $MIU(b)$ ,  $MIU(c)$  and  $MIU(e)$ . Hence,

(*bce*) is not a HTWUI, and thus the itemset (*bced*) and its supersets would be discarded according to the TWDC property. But it can be observed that  $TWU(bced) = 50$ , which is equal to  $MIU(bced) = 50$ . But as shown in Table 2, it can be seen that itemset (*bced*) is actually a HUI. It is thus incorrect to discard the supersets of (*bce*) based on the TWDC property since (*bced*) would not be generated. Therefore, if a  $k$ -itemset  $X^k$  is a HTWUI (i.e.,  $TWU(X^k) \geq MIU(X^k)$ ), we cannot ensure that any subset  $X^{k-1}$  of  $X^k$  is also a HTWUI (because  $MIU(X^{k-1}) \geq MIU(X^k)$ ). Thus, using this property to prune the search space may fail to discover the complete set of HUIs. To address this limitation, the *Sorted Downward Closure (SDC) property* was proposed in [9].

**Theorem 1 (Sorted Downward Closure Property, SDC Property).**

Assume that items in itemsets are sorted by ascending order of  $mu$  values. Given any itemset  $X^k = \{i_1, i_2, \dots, i_k\}$  of length  $k$ , and another itemset  $X^{k-1} = \{i_1, i_2, \dots, i_{k-1}\}$  such that  $X^{k-1} \subseteq X^k$ . If  $X^k$  is a HTWUI then  $X^{k-1}$  is also a HTWUI [9].

*Proof.* Since  $X^{k-1} \subseteq X^k$ , the following relationships hold:

- (1) By Definition 2, we have that  $MIU(X^{k-1}) = \min\{mu(i_1), mu(i_2), \dots, mu(i_{k-1})\}$ , and  $MIU(X^k) = \min\{mu(i_1), mu(i_2), \dots, mu(i_k)\}$ . Since  $\{i_1, i_2, \dots, i_k\}$  is sorted according to the total order  $\prec$ ,  $MIU(X^k) = MIU(X^{k-1}) = mu(i_1)$ .
- (2) Thus,  $TWU(X^k) = \sum_{X^k \subseteq T_q \wedge T_q \in D} tu(T_q) \leq \sum_{X^{k-1} \subseteq T_q \wedge T_q \in D} tu(T_q) = TWU(X^{k-1})$ . Therefore, if  $X^k$  is a HTWUI (i.e.,  $TWU(X^k) \geq mu(i_1)$ ), any subset  $X^{k-1}$  of  $X^k$  is also a HTWUI.

Although the *sorted downward closure (SDC) property* guarantees the anti-monotonicity for HTWUIs, some HUIs would still be missed if items that are HTWUIs are determined using their  $MIU(X)$  values. To address this problem, the concept of least minimum utility value ( $LMU$ ) was developed to guarantee deriving all HUIs when using multiple minimum utility thresholds [9].

**Definition 14 (Least Minimum Utility Value,  $LMU$ ).** The least minimum utility value ( $LMU$ ) is defined as the smallest value in the  $MMU$ -table, that is:

$$LMU = \min\{mu(i_1), mu(i_2), \dots, mu(i_m)\}, \quad (10)$$

where  $m$  is the total number of items in the database.

For example, the  $LMU$  of the given example is calculated as:  $\min\{mu(a), mu(b), mu(c), mu(d), mu(e)\} = \min\{56, 65, 53, 50, 70\} = 50$ .

## 4.2 Proposed Conditional Downward Closure (CDC) and Global Downward Closure (GDC) Properties

**Lemma 2.** The  $MIU$  value of a node/pattern in the  $MIU$ -tree is equal to that of any of its child nodes (extension nodes).

*Proof.* Assume that  $X^{k-1}$  is a node representing an itemset  $X$  in the MIU-tree, and that  $X^k$  is any of its child nodes (extensions). By definition, we have that  $MIU(X^{k-1}) = \min\{\mu(i_1), \mu(i_2), \dots, \mu(i_{k-1})\}$ , and  $MIU(X^k) = \min\{\mu(i_1), \mu(i_2), \dots, \mu(i_k)\}$ . Since  $\{i_1, i_2, \dots, i_k\}$  is sorted by ascending order of  $\mu$  values, it can be proven that:  $MIU(X^k) = MIU(X^{k-1}) = \mu(i_1)$ . Thus, the  $MIU$  value of a node in the MIU-tree is always equal to the  $MIU$  of any of its child nodes.

**Lemma 3.** *The support of a node in the MIU-tree is no less than the support of any of its child nodes (extension nodes).*

*Proof.* Since the Set-enumeration MIU-tree is a prefix tree, the relationship of the support of  $X^k$  and  $X^{k-1}$  can be proven to be  $sup(X^k) \leq sup(X^{k-1})$ .

**Theorem 2 (HUIs  $\subseteq$  HTWUIs).** *Assume that 1-itemsets having a TWU lower than LMU are discarded and that the total order  $\prec$  is applied. We have that HUIs  $\subseteq$  HTWUIs, which indicates that if an itemset is not a HTWUI, then it is not a HUI. Moreover, none of its extensions are HTWUIs or HUIs.*

*Proof.* Let  $X^k$  be an itemset such that  $X^{k-1}$  is a subset of  $X^k$ .

- (1) We have that  $TWU(X^1) \leq LMU$  and  $MIU(X^k) \geq LMU$ .
- (2) Since items are sorted by ascending order of  $\mu$  values,  $TWU(X^{k-1}) \geq TWU(X^k)$  and  $MIU(X^{k-1}) = MIU(X^k) = \min\{\mu(i_1), \mu(i_2), \dots, \mu(i_m)\} = \mu(i_1)$ .
- (3)  $u(X) = \sum_{X \subseteq T_q \wedge T_q \in D} u(X, T_q) \leq \sum_{X \subseteq T_q \wedge T_q \in D} tu(T_q) = TWU(X)$ .

Thus, if  $X^{k-1}$  is not a HTWUI and  $TWU(X^{k-1}) < \mu(i_1)$ , none of its supersets are HUIs.

**Lemma 4.** *The TWU of any node in the Set-enumeration MIU-tree is no less than the sum of all the actual utilities of any one of its descendant nodes, but not the MIU of its descendant nodes.*

*Proof.* Let  $X^{k-1}$  be a node in the MIU-tree, and  $X^k$  be a children (extension) of  $X^{k-1}$ . According to Theorem 1 and Lemma 1, we can get  $TWU(X^{k-1}) \geq TWU(X^k)$  and the relationship between  $MIU$  values. Thus, the lemma holds.

**Theorem 3 (Global Downward Closure Property, GDC Property).** *In the designed MIU-tree, if the TWU of a tree node  $X$  is less than the LMU,  $X$  is not a HUI, and all its supersets (not only its child nodes, but all nodes containing  $X$ ) are also not considered as HUIs.*

*Proof.* According to Lemma 2 and Theorem 2, this theorem holds.

This theorem ensures that by discarding itemsets with a TWU less than  $LMU$ , and their extensions, no HUIs are missed. Thus, the designed *global downward closure (GDC) property* and the  $LMU$  guarantee the **completeness** and **correctness** of the proposed HIMU algorithm, when pruning the search space.



In the past, a structure named utility-list was proposed to keep information from transactions in memory to directly mine HUIs [10]. The utility-list structure is efficient and is thus adopted in the proposed HIMU algorithm to store the required information about itemsets, as shown in Fig. 2. The reader can refer to [8,10] for details about the utility-list structure, and the *iu* and *ru* values stored in utility-lists.

(d)		
<i>tid</i>	<i>iu</i>	<i>ru</i>
1	9	8
2	9	18
4	27	15
5	18	42
6	18	24
7	9	41
9	18	5
10	9	14

(c)		
<i>tid</i>	<i>iu</i>	<i>ru</i>
1	2	6
3	5	12
4	3	12
7	2	39
8	3	12
9	2	3
10	2	12

(a)		
<i>tid</i>	<i>iu</i>	<i>ru</i>
1	6	0
2	12	6
4	6	6
8	12	0
10	12	0

(b)		
<i>tid</i>	<i>iu</i>	<i>ru</i>
3	12	0
5	36	6
6	24	0
7	36	3

(e)		
<i>tid</i>	<i>iu</i>	<i>ru</i>
2	6	0
4	6	0
5	6	0
7	3	0
9	3	0

Fig. 2. Constructed utility-lists of 1-itemsets in the running example.

**Definition 15.** For an itemset  $X$ ,  $X.IU$  and  $X.RU$  are respectively the sum of *iu* values and the sum of *ru* values in the utility-list of  $X$ , that is:

$$X.IU = \sum_{X \subseteq T_q \wedge T_q \in D} X.iu(T_q); X.RU = \sum_{X \subseteq T_q \wedge T_q \in D} X.ru(T_q).$$

**Strategy 1.** When traversing the MIU-tree using a depth-first search, if the TWU of a node  $X$  based on its utility-list is less than the LMU, then none of the supersets of node  $X$  (note that here supersets contains not only descendant nodes of  $X$ , but also other nodes having  $X$  as subset) are HUIs.

**Theorem 4 (Conditional Downward Closure Property, CDC Property).**

For any node  $X$  in the MIU-tree, the sum of  $X.IU$  and  $X.RU$  in the utility-list of  $X$  is no less than the utility of any one of its descendant nodes (extensions). Thus this sum is anti-monotonic and allows pruning itemsets in the MIU-tree.

*Proof.* Let  $X^{k-1}$  be a  $(k-1)$ -itemset, and  $X^k$  be a  $(k)$ -itemset that is an extension of  $X^{k-1}$ . Assume that  $X^k$  is a children of  $X^{k-1}$  in the MIU-tree, meaning that  $X^{k-1}$  is a prefix of  $X^k$ . Let the set of items in  $X^k$  but not in  $X^{k-1}$  be denoted as  $(X^k - X^{k-1}) = (X^k \setminus X^{k-1})$ , and the set of all the items appearing after  $X^k$  in transaction  $T$  is denoted as  $T/X^k$ . For any transaction  $X^k \subseteq T_q$ :

$$\begin{aligned} \therefore X^{k-1} \subset X^k \subseteq T_q &\Rightarrow (X^k \setminus X^{k-1}) \subseteq (T_q \setminus X^{k-1}). \\ \therefore \text{In each } T_q, X^k.iu &= X^{k-1}.iu + (X^k \setminus X^{k-1}).iu = X^{k-1}.iu + \sum_{z \in (X^k \setminus X^{k-1})} z.iu \\ \therefore X^k.iu &\leq X^{k-1}.iu + \sum_{z \in (T_q/X^{k-1})} z.iu = X^{k-1}.iu + X^{k-1}.ru \\ \therefore \text{In each } T_q, X^k.iu &\leq X^{k-1}.iu + X^{k-1}.ru \\ \therefore X^{k-1} \subset X^k &\Rightarrow X^k.tids \subseteq X^{k-1}.tids \\ \therefore \text{in } D, X^k.IU &= \sum_{T_q \in X^k.tids} X^k.iu \leq \sum_{T_q \in X^k.tids} (X^{k-1}.iu + X^{k-1}.ru) \end{aligned}$$

$$\leq \sum_{T_q \in X^{k-1}.tids} (X^{k-1}.iu + X^{k-1}.ru) = X^{k-1}.IU + X^{k-1}.RU$$

$\therefore$  in  $D$ ,  $X^k.IU \leq X^{k-1}.IU + X^{k-1}.RU$

Thus, the sum of the utilities of  $X^k$  in  $D$  is no greater than  $(X^{k-1}.IU + X^{k-1}.RU)$  of  $X^{k-1}$  in  $D$ .

**Strategy 2.** *When traversing the MIU-tree using a depth-first search, if the sum of  $X.IU$  and  $X.RU$  in the utility-list of an itemset  $X$  is less than  $MIU(X)$ , then none of the descendant nodes (extensions) of node  $X$  is a HUI since the actual utilities of these extensions will be less than  $MIU(X)$ .*

In the running example, assume that the node ( $e$ ) has  $TWU(e) < LMU$ . Then the visited nodes, pruned nodes, and the skipped nodes are respectively shown in Fig. 1 (right) when applying the **Strategy 1**. And the **Strategy 2** is used as a conditional strategy to prune all extensions of an unpromising node early.

### 4.3 Estimated Utility Co-occurrence Pruning Strategy

In this section, we extend the Estimated Utility Co-occurrence Pruning (EUCP) strategy [8], in the proposed algorithm, to provide an additional way of pruning unpromising itemsets early with multiple minimum utility thresholds.

**Theorem 5.** *Without loss of generality, assume that items in itemsets are sorted by ascending order of  $mu$  values. If an itemset  $X$  contains a 2-itemset  $X$  that is not a HTWUI, then any  $k$ -itemset  $X^k$  ( $k \geq 3$ ) that is a (transitive) extension of  $X$  is not a HTWUI or HUI.*

*Proof.* Let  $X$  be a 2-itemset and  $X^k$  be a  $k$ -itemset ( $k \geq 3$ ) that is a (transitive) extension of  $X$ . According to the *GDC* property and because  $TWU(X^k) \leq TWU(X^{k-1})$ , if a 2-itemset is not a HTWUI, then any  $k$ -itemset ( $k \geq 3$ ), which is an extension of  $X$  is not a HTWUI or HUI.

As mentioned above, not all supersets of a non HTWUI should be pruned but only those having a *MIU* value greater than the *MIU* value of this non HTWUI (w.r.t. the extensions of this non HTWUI having higher *MIU* values). Thus, using the proposed *GDC* property with the designed total order  $\prec$ , the **completeness** and **correctness** of the enhanced algorithm named  $HIMU_{EUCP}$  is preserved by extending the EUCP strategy. Note that the *TWU* values of all 2-itemsets are stored in a structure called estimated utility co-occurrence structure (*EUCS*) [8].

**Strategy 3 (EUCP Strategy).** *When traversing the MIU-tree using a depth-first search, if the *TWU* value of a 2-itemset  $X$  is less than the *MIU* value of  $X$  according to the *EUCS*, then  $X$  is not a HTWUI; and any  $k$ -itemset which is an extension of  $X$  will not be a HTWUI or HUI, and they can be pruned directly.*

#### 4.4 Procedure of the HIMU Algorithm and the Enhanced Algorithm

Note that the proposed enhanced  $\text{HIMU}_{\text{EUCP}}$  algorithm is similar to the baseline HIMU algorithm. The difference is that the  $\text{EUCS}$  needs to be constructed initially during the second database scan. Moreover, the mining procedure for deriving HUIs is modified to verify pruning Strategy 3 for each generated itemset. Due to the page limitation, only the details of the  $\text{HIMU}_{\text{EUCP}}$  algorithm are provided.

**Input:**  $D$ ;  $p\text{-table}$ ;  $\text{MMU-table} = \{mu(i_1), mu(i_2), \dots, mu(i_m)\}$ .

**Output:** The set of complete high-utility itemsets (HUIs).

- 1  $i.UL \leftarrow \emptyset, D.UL \leftarrow \emptyset, \text{EUCS} \leftarrow \emptyset$ ;
- 2 calculate the  $LMU$  in the  $\text{MMU-table}$ ;
- 3 scan  $D$  to calculate the  $TWU(i)$  value of each item  $i \in I$ ;
- 4 find  $I^* \leftarrow \{i \in I | TWU(i) \geq LMU\}$ , w.r.t.  $\text{HTWUI}^1$ ;
- 5 sort  $I^*$  according to the designed total order  $\prec$  (ascending order of  $mu$  values);
- 6 scan  $D$  to construct the utility-list  $i.UL$  of each item  $i \in I^*$  and build the  $\text{EUCS}$ ;
- 7 call **HUI-Search**( $\phi, I^*, \text{MMU-table}, \text{EUCS}$ );
- 8 return  $\text{HUIs}$ ;

**Algorithm 1.** The  $\text{HIMU}_{\text{EUCP}}$  algorithm

**Input:**  $X$ ,  $\text{extensionsOf}X$ ,  $\text{MMU-table}$ ,  $\text{EUCS}$ .

**Output:** The complete set of HUIs.

- 1 **for** each itemset  $X_a \in \text{extensionsOf}X$  **do**
- 2     obtain the  $X_a.IU$  and  $X_a.RU$  values from the built  $X_a.UL$ ;
- 3     **if**  $X_a.IU \geq MIU(X_a)$  **then**
- 4          $\text{HUIs} \leftarrow \text{HUIs} \cup X_a$ ;
- 5     **if** ( $X_a.IU + X_a.RU \geq MIU(X_a)$ ) **then**
- 6          $\text{extensionsOf}X_a \leftarrow \emptyset$ ;
- 7         **for** each itemset  $X_b \in \text{extensionsOf}X$  such that  $X_b$  after  $X_a$  **do**
- 8             **if**  $\exists TWU(a, b) \in \text{EUCS} \wedge TWU(a, b) \geq MIU(X_a)$  **then**
- 9                  $X_{ab} \leftarrow X_a \cup X_b$ ;
- 10                  $X_{ab}.UL \leftarrow \text{construct}(X, X_a, X_b)$ ;
- 11                 **if**  $X_{ab}.UL \neq \emptyset$  **then**
- 12                      $\text{extensionsOf}X_a \leftarrow \text{extensionsOf}X_a \cup X_{ab}.UL$ ;
- 13         **call** **HUI-Search**( $X_a, \text{extensionsOf}X_a, MIU(X_a), \text{EUCS}$ );
- 14 **return**  $\text{HUIs}$

**Algorithm 2.** The HUI-Search Procedure

As shown in Algorithm 1, the  $\text{HIMU}_{\text{EUCP}}$  algorithm first sets  $i.UL$ ,  $D.UL$  and  $\text{EUCS}$  to the empty set (Line 1), and calculates the  $LMU$  in the  $\text{MMU-table}$  (Line 2). Then, it scans the database to calculate the  $TWU(i)$  value of each item  $i \in I$  (Line 3), and then find the potential 1-itemsets which may be

HUIs such that  $TWU(i) \geq LMU(I^* \subseteq HTWUI^1)$  (Line 4). After sorting  $I^*$  by  $\prec$  (ascending order of  $mu$  values), the algorithm scans  $D$  again to construct the utility-list of each item  $i \in I^*$  and build the *EUCS* (Lines 5 to 6). It is important to notice that only the designed order  $\prec$  can guarantee the completeness of HIMU, as previously explained. The utility-list of each item  $i \in I^*$  is recursively processed by the depth-first search HUI-Search procedure (Line 7). This latter procedure (cf. Algorithm 2), checks if each 1-extension  $X_a$  of an itemset  $X$  is a HUI (Lines 2 to 4). Two conditions are then checked to determine whether its child nodes should be considered by the depth-first search (Lines 5 to 12). If an itemset is regarded as a potential HUI, the *Construct*( $X, X_a, X_b$ ) procedure (see [10] for details) is applied to construct the utility-lists of all 1-extensions of  $X_a$  (w.r.t. *extensionsOf* $X_a$ ) (Lines 9 to 12). Notice that each extension  $X_{ab}$  is a 1-extension of itemset  $X_a$ , and is added to the set *extensionsOf* $X_a$  for the later depth-first search (Line 13). The HUI-Search procedure then is recursively called to mine HUIs (Line 13).

## 5 Experimental Evaluation

The performance of the proposed HIMU and HIMU<sub>EUCP</sub> algorithms was evaluated on two real-world datasets, foodmart [4] and mushroom [1]. The foodmart dataset contains customer transactions from an anonymous chain store, and is provided with Microsoft SQL Server. It contains 21,556 transactions and 1,559 distinct items. The mushroom dataset is dense. It has 8,124 transactions and 120 distinct items, and an average transaction length of 23 items. The foodmart dataset contains real utility values, while a simulation model [13] was developed to generate the quantities and profit values of items in transactions for the mushroom dataset, by choosing random values respectively in the [1, 5] and [1, 1000] intervals.

The performance of the designed algorithms was also compared with the state-of-the-art HUI-MMU and HUI-MMU<sub>TID</sub> algorithms [9]. To perform a fair comparison, all algorithms were implemented in Java and executed on a computer having an Intel Core2 Duo 2.8 GHz processor and 4GB of main memory, running the 64 bit Microsoft Windows 7 operating system. Moreover, a method to automatically set the  $mu$  value of each item was adopted, described in the HUI-MMU algorithm [9]:  $mu(i_j) = \max[\beta \times pr(i_j), GLMU]$ , where  $\beta$  is a constant used to set the  $mu$  values of items as a function of their unit profit values. To ensure randomness and diversity in the experiments,  $\beta$  was set in the [1, 100] interval for the foodmart dataset, and in the [1000, 10000] for mushroom. The parameter  $GLMU$  is a user-specified global least minimum utility value, and  $pr(i_j)$  is the external utility of an item  $pr(i_j)$ . Note that if  $\beta$  is set to zero, then a single minimum utility value  $GLMU$  will be used for all items, and this will be equivalent to traditional HUIM.

## 5.1 Execution Time

In the conducted experiments, the parameter  $\beta$  was randomly set to a fixed number of items. Figure 3 shows the runtime of the algorithms under various  $GLMU$  with a fixed  $\beta$  within an interval, and under various  $\beta$  with a fixed  $GLMU$  for different datasets. In Fig. 3, it can be seen that the HIMU and the improved HIMU<sub>EUCP</sub> algorithms perform well compared to the HUI-MMU, HUI-MMU<sub>TID</sub> algorithms under various  $GLMU$  with a fixed  $\beta$ , and under various  $\beta$  with a fixed  $GLMU$ . Moreover, the two MIU-tree-based algorithms are generally up to almost one or two orders of magnitude faster than the level-wise HUI-MMU and HUI-MMU<sub>TID</sub> algorithms. HIMU<sub>EUCP</sub> is faster than the HIMU algorithm on mushroom but not on foodmart, by adopting the EUCP strategy, which is used to avoid join operations for forming the utility-lists of unpromising itemsets. This indicates that the generate-and-test approach has worse performance than the proposed depth-first search approach that utilizes the vertical utility-list structure and additional pruning strategies. The gap between the previous approaches and the proposed MIU-tree-based algorithms becomes large when  $GLMU$  and  $\beta$  are decreased. As shown in Fig. 3(a) and (c), HIMU<sub>EUCP</sub> performs slightly worse than HIMU. The reason is that for the very sparse foodmart dataset, with an average transaction length of 4.4, many unpromising candidates can be directly pruned by the redefined HTWUI and SDC properties, and it is unnecessary to apply the EUCP strategy for pruning unpromising itemsets, and thus construct the *EUCS*. Furthermore, when  $\beta$  is increased, the HIMU and HIMU<sub>EUCP</sub> algorithms take less time to find the HUIs. The reason is that when  $\beta$  is set to large values, the actual minimum utility threshold of each item is also set to larger

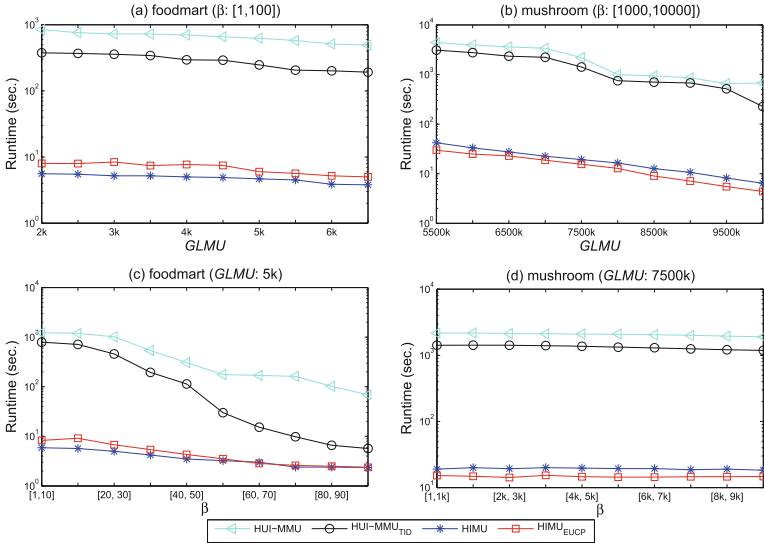


Fig. 3. Runtime performance.

values based on the presented equation. Hence, fewer HUIs and HTWUIs are pruned by the pruning conditions, and the execution time becomes smaller. In summary, the two proposed algorithms considerably outperform the state-of-the-art HUI-MMU and HUI-MMU<sub>TID</sub> algorithms.

## 5.2 Effect of Pruning Strategies

We also evaluated the effectiveness of the EUCP strategy for pruning unpromising itemsets. The number of itemsets (nodes) visited by the HIMU and HIMU<sub>EUCP</sub> algorithms are named  $N_2$  and  $N_3$ , respectively. Moreover, the number of itemsets generated by combining pairs for determining HTWUIs in HUI-MMU and HUI-MMU<sub>TID</sub> is denoted as  $N_1$ . Results are shown in Fig. 4. It can be observed that relationship  $N_3 \leq N_2$  holds for all datasets no matter how *GLMU* is set, for a fixed  $\beta$  or under various  $\beta$  with a fixed *GLMU*. Especially, the node reduction obtained by adopting the EUCP strategy in the enhanced algorithm is huge, as shown in  $N_3 \leq N_2$ . Besides,  $N_1$  is less than  $N_3$  and  $N_2$  for foodmart, but larger on mushroom. It indicates that the search space (in terms of visited nodes in the Set-enumeration MIU-tree) of HIMU may be huge if the effective EUCP pruning strategy is not applied for pruning the search space.

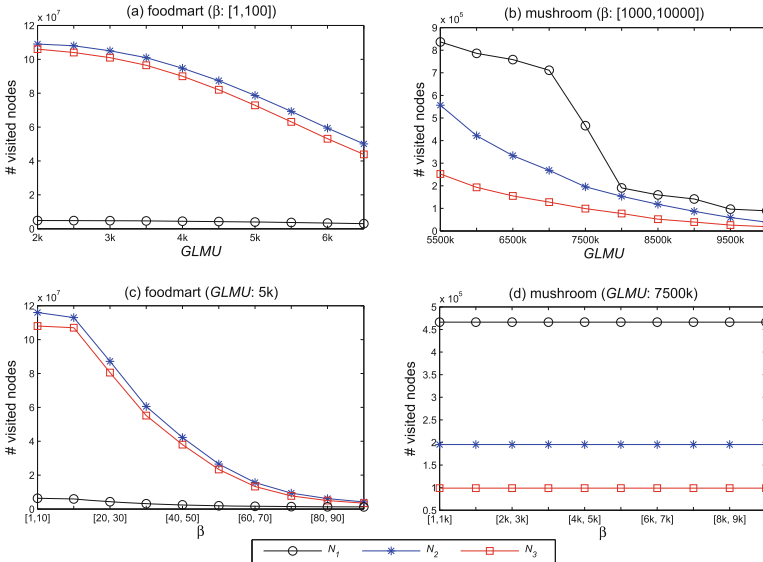


Fig. 4. Number of visited nodes (patterns).

## 5.3 Memory Consumption

We also assessed the memory consumption of the compared algorithms. Memory measurements were done using the standard Java API. Note that the peak

memory consumption of each algorithm was recorded for all datasets. Results are shown in Fig. 5. In this figure, it can be clearly seen that the proposed HIMU algorithms require less memory than the state-of-the-art HUI-MMU and HUI-MMU<sub>TID</sub> algorithms for various parameters on the two datasets, and by up to 3,000 times on mushroom. Moreover, the HIMU and HIMU<sub>EUCP</sub> algorithms require nearly constant memory under various parameter values for the two datasets. The memory usage of the level-wise algorithms dramatically increases when *GLMU* or  $\beta$  are decreased, while the memory usage of the proposed algorithms remain stable. The reason is the same as above. This result is reasonable since the two MIU-tree-based algorithms can quickly traverse the MIU-tree without generating candidates and easily prune unpromising itemsets using the sum of utilities and remaining utilities. Furthermore, the utility-list structure is adopted as a vertical compact structure to store information about itemsets. Thus, less memory is consumed.

#### 5.4 Scalability Analysis

Figure 6 compares the scalability of the algorithms on synthetic data T10I4N4KD|X|K where the transaction count (K) was varied from 100K to 500K,  $GLMU = 1,000,000$  and  $\beta$  was varied from 1000 to 10000. It can be seen that the designed algorithms scale well with respect to dataset size and that HIMU<sub>EUCP</sub> scales better than HIMU. When the dataset size is increased, HIMU<sub>EUCP</sub> becomes increasingly faster than the other algorithms thanks to the EUCP strategy. HIMU<sub>EUCP</sub> consumes more memory than HIMU but less than the two level-wise algorithms because it uses the additional *EUCS* to store TWU values of all 2-itemsets (see Fig. 6(b)). From Fig. 6(c), it can also be seen that

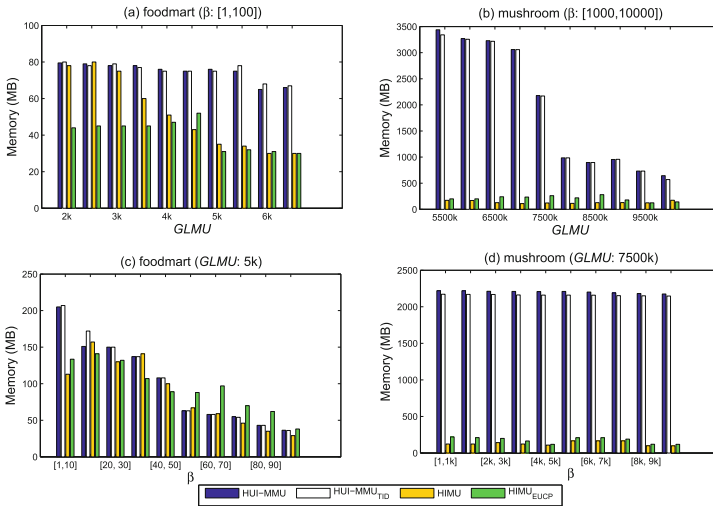


Fig. 5. Memory performance.

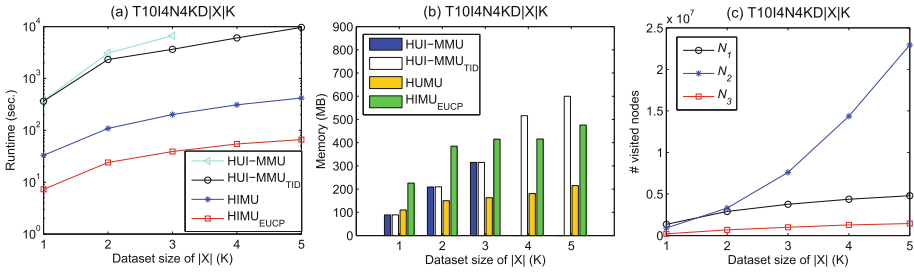


Fig. 6. Scalability of the compared algorithms.

the number of nodes  $N_3$  remains much smaller than  $N_2$ . Thus, by utilizing the EUCP strategy, the actual search space of the HIMU<sub>EUCP</sub> algorithm is reduced compared to the baseline HIMU algorithm, and it can be concluded that the improved algorithm is acceptable and efficient.

## 6 Conclusion

In this paper, a novel algorithm named HIMU was presented to discover high-utility itemsets with multiple minimum utility thresholds. A compact Multiple Itemset Utility Set-enumeration tree (MIU-tree) was designed for mining HUIs without candidate generation. Besides, the *global and conditional downward closure (GDC and CDC) properties* were proposed to guarantee the global and partial anti-monotonicity for mining HUIs. Pruning conditions are also incorporated in the proposed algorithms to reduce the search space, and an efficient compact utility-list structure is used to obtain information about any itemset from its prefix itemsets in the designed MIU-tree. An experimental evaluation against the state-of-the-art HUI-MMU and HUI-MMU<sub>TID</sub> algorithms on two real-world datasets shows that the two proposed algorithms are highly efficient and scalable.

**Acknowledgment.** This research was partially supported by the National Natural Science Foundation of China (NSFC) under Grant No. 61503092, and by the Tencent Project under grant CCF-TencentRAGR20140114.

## References

1. Frequent itemset mining dataset repository. <http://fimi.ua.ac.be/data/>
2. Agrawal, R., Imielinski, T., Swami, A.: Database mining: a performance perspective. *IEEE Trans. Knowl. Data Eng.* **5**(6), 914–925 (1993)
3. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: *The International Conference on Very Large Data Bases*, pp. 487–499 (1994)
4. Microsoft. Example database foodmart of Microsoft analysis services. <http://www.Almaden.ibm.com/cs/quest/syndata.html>



5. Ahmed, C.F., Tanbeer, S.K., Jeong, B.S., Le, Y.K.: Efficient tree structures for high utility pattern mining in incremental databases. *IEEE Trans. Knowl. Data Eng.* **21**(12), 1708–1721 (2009)
6. Chan, R., Yang, Q., Shen, Y.D.: Mining high utility itemsets. In: *The International Conference on Data Mining*, pp. 19–26 (2003)
7. 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)
8. 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)
9. Lin, J.C.W., Gan, W., Fournier-Viger, P., Hong, T.P.: Mining high-utility itemsets with multiple minimum utility thresholds. In: *ACM International Conference on Computer Science & Software Engineering*, pp. 9–17 (2015)
10. Liu, M., Qu, J.: Mining high utility itemsets without candidate generation. In: *ACM International Conference on Information and Knowledge Management*, pp. 55–64 (2012)
11. 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)
12. 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)
13. Tseng, V.S., Wu, C.W., Shie, B.E., Yu, P.S.: UP-growth: an efficient algorithm for high utility itemset mining. In: *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 253–262 (2010)
14. 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)
15. Hu, Y.H., Chen, Y.L.: Mining association rules with multiple minimum supports: a new mining algorithm and a support tuning mechanism. *Decis. Support Syst.* **42**(1), 1–24 (2006)
16. Yao, H., Hamilton, J., Butz, C.J.: A foundational approach to mining itemset utilities from databases. In: *SIAM International Conference on Data Mining*, pp. 211–225 (2004)