Mining High Utility Itemsets with Regular Occurrence

Komate Amphawan[1,*], Philippe Lenca[2], Anuchit Jitpattanakul[3] & Athasit Surarerks[4]

[1]Burapha University, Computational Innovation Laboratory, 20131 Chonburi, Thailand
[2]Institut Telecom, Telecom Bretagne, UMR CNRS 3192 Lab-STICC, France
[3]Faculty of Applied Science, KNUTNB, 10800 Bangkok, Thailand
[4]Chulalongkorn University, ELITE Laboratory, 10330 Bangkok, Thailand
*E-mail: komate@gmail.com

Abstract. High utility itemset mining (HUIM) plays an important role in the data mining community and in a wide range of applications. For example, in retail business it is used for finding sets of sold products that give high profit, low cost, etc. These itemsets can help improve marketing strategies, make promotions/ advertisements, etc. However, since HUIM only considers utility values of items/itemsets, it may not be sufficient to observe product-buying behavior of customers such as information related to “regular purchases of sets of products having a high profit margin”. To address this issue, the occurrence behavior of itemsets (in the term of regularity) simultaneously with their utility values was investigated. Then, the problem of mining high utility itemsets with regular occurrence (MHUIR) to find sets of co-occurrence items with high utility values and regular occurrence in a database was considered. An efficient single-pass algorithm, called MHUIRA, was introduced. A new modified utility-list structure, called NUL, was designed to efficiently maintain utility values and occurrence information and to increase the efficiency of computing the utility of itemsets. Experimental studies on real and synthetic datasets and complexity analyses are provided to show the efficiency of MHUIRA combined with NUL in terms of time and space usage for mining interesting itemsets based on regularity and utility constraints.

Keywords: *association rule mining; data mining; high utility itemsets; occurrence behavior; regularity constraint; utility-list structure.*

1 Introduction

Association rule mining (ARM) [1,2] is a fundamental task of data mining and data analysis. It aims to discover a relationship between objects or events, which is expressed in the form of a $X \rightarrow Y$ rule. For example, from purchasing data of a retail business, ARM may discover the rule “$Beer \rightarrow Diaper$ [s: 30%, f: 60%]” which expresses buying behavior of customers, i.e. 30% of customers bought $Beer$ simultaneously with $Diaper$ and 60% of customers who bought $Beer$ also bought $Diaper$ at the same time. ARM can be applied in several areas such as retail marketing, web clickstream analysis and DNA analysis. ARM consists

Received October 30th, 2015, Revised May 9th, 2016, Accepted for publication May 31st, 2016.
 ISSN: 2337-5787, DOI: 10.5614/itbj.ict.res.appl.2016.10.2.5

of two main steps: (i) frequent itemset mining – finding sets of items with the frequency of occurrence satisfying a user-specified frequency (support) threshold; and (ii) rule generation – generating interesting rules from frequent itemsets of Step (i) that meet a user-given confidence threshold.

From the two steps mentioned above, frequent itemset mining is much more attractive than rule generation due to the explosive growth of the search space. Therefore, researchers have mostly focused on improving performance of frequent itemset mining algorithms. In addition, frequent itemset mining only considers the frequency of occurrence as a criterion to measure interestingness of itemsets, which may not be sufficient to track interesting occurrence behavior of itemsets. Thus, Tanbeer, *et al.* [3] proposed to add a regularity constraint to discover sets of items that frequently and regularly occur in a collection of data. This can help to find out about the regularity of product purchase, which can in turn help to manage inventory, design a marketing strategy, etc. However, the consideration of only frequency and/or regularity of occurrence cannot reflect the importance and/or utilit of items.

Chan, *et al.* [4] proposed to consider the importance of items, such as profit, cost and other user-defined factors as well as the quantity of occurrence (e.g. units of a product bought by each customer). Based on these two components, an itemset is called a high utility itemset (*HUI*) if its utility (i.e. unit profit × quantity of occurrence) is no less than a user-assigned minimum utility threshold (σ_{u}); otherwise, it is called a low utility itemset (*LUI*). Mining *HUI* has a wide range of applications, such as crossmarketing in retail, web clickstream analysis, biomedical applications and mobile commerce.

Although mining *HUI* can discover interesting sets of items with high utility values, it may not be sufficient to analyze interesting buying behaviors of customers. For example, as shown in Table 1 (containing per-unit-profits of items) and Table 2 (a transactional database with quantity of occurrence), it can be seen that item 'h' occurs in transactions t_2 and t_6 with 20 and 1 pieces bought with a profit of 15\$ per unit sold. Thus, 'h' tends to be a high utility itemset due to its utility of 315\$ (i.e. $(20 \times 15) + (1 \times 15)$). However, '$h$' was bought only twice and one purchase was a large quantity (compared with the other one). On the other hand, we can observe that item 'd' has the highest profit per unit (i.e. 30\$), which is really high in comparison with the other items. Then, item 'd' and its supersets might be high utility itemsets even if 'd' has only a small quantity and frequency of occurrence. Moreover, there are cases where the utility value alone cannot indicate interesting itemsets (patterns). Hence, it is better to consider the utility of itemsets along with their occurrence behavior.

Table 1 External utility of items (items' unit utility).

a	**b**	**c**	**d**	**e**	**f**	**g**	**h**
3	2	1	30	5	3	4	15

Table 2 Transactional database with internal utilities.

tid	**Items (internal utility)**
1	a(3), c(8), d(2), e(1)
2	b(5), f(3), g(5), h(20)
3	a(2), c(4), d(1)
4	c(5), e(1), f(1)
5	a(2), b(3), c(1), f(4)
6	d(1), g(5), h(1)
7	a(5), b(1), d(1), e(2)
8	a(3), b(1), c(4), d(1), e(1)

To address the abovementioned issues, we propose to consider a regularity constraint together with high utility itemset mining (also called mining high utility itemsets with regular occurrence, *MHUIR*) as in [5]. These itemsets can produce information related to "regular purchases of sets of products with a high profit margin" and can help business know customers' demand, manage inventory, develop new promotions/marketing strategies, and so on. To mine high utility itemsets with regular occurrence (*HUIRs*), an efficient single-pass algorithm named *MHUIRA* is proposed. *MHUIRA* avoids repeatedly scanning the database by employing a simple list to maintain the utility information of each transaction. Furthermore, the concepts of transaction-weighted utility [6], tight over-estimated utility [7] are applied early – in order to remove low utility itemsets out of consideration – and a utility list, called *UL*, is applied for maintaining utility and occurrence information of each itemset (as in [5]). Moreover, a new modified utility list structure (also called *NUL* for short) was designed in this extended version to increase the efficiency of the computing of the itemsets. Experimental studies and a complexity analysis are provided to show the efficiency and effectiveness of *MHUIRA with UL* and *NUL* in terms of runtime, memory usage and number of discovered itemsets.

The rest of this paper is organized as follows. Section 2 briefly discusses related works. Section 3 describes the basic notation for discovering *HUIR*s. The concept of utility list, new modified utility list and the *MHUIRA* algorithm are detailed in Section 4. Experimental results and the complexity analysis are discussed in Section 5. Section 6 gives the conclusion of this paper.

2 Related Works

In this section, high utility itemset mining (*HUIM*) and frequent-regular itemset mining (*FRIM*) are reviewed briefly.

2.1 High Utility Itemset Mining (HUIM)

Since Chan [4] first proposed high utility itemset mining, many studies have been conducted to improve performance of the mining process. Liu [6] proposed a two-phase algorithm with an overestimated utility strategy named transaction-weighted utility (*TWU*) used for pruning the search space. A *TWU* of an itemset is an upper bound of its utility, which keeps the downward closure property [2]. Based on *TWU*, Lin [8] proposed the high utility pattern tree (*HUP-tree*) algorithm for mining high utility itemsets without candidate generation. In [9], a pattern growth approach named *UP-growth* was proposed for mining high utility itemsets within two scans of a database. *Up-growth* applies four effective strategies, i.e. *DGU*, *DGN*, *DLU* and *DLN* to prune candidates during the mining process. Liu [7] proposed the HUI-miner algorithm with a tight overestimated utility strategy, which can estimate utility values close to the actual utility of itemsets. Tight overestimated utility is used for reducing the search space.

High utility itemset mining has also been extended to several other aspects. Wu [10] proposed three efficient algorithms to mine *closed+ high utility itemsets,* a concise representation of high utility itemsets. *HUIM* on incremental and modification databases and on data streams is proposed in [11-14]. Negative unit profits of items are addressed in [15,16]. To avoid difficulties in setting a proper utility threshold, in [17-19] attempts were done to mine a set of *k* itemsets with the highest utility. Podpecan [20] and Sugunadevi [21] proposed to discover high utility-frequent itemsets based on consideration of utility and frequency of occurrence. High utility-frequent itemsets can express frequently sold products with high utility. However, this approach does not observe occurrence behavior in other aspects that may give significant and interesting information for further decision-making.

2.2 Frequent-Regular Itemset Mining (FRIM)

Tanbeer [3] has proposed the problem of frequent-regular itemset mining (*FRIM*) in order to observe occurrence behavior of itemsets in terms of frequency and regularity of occurrence. A set of itemsets with high frequency and regular occurrence that meets user-given support and regular thresholds is generated. *FRIM* can be applied in a wide range of applications, such as genetic and medical data analysis [22-24], manufacturing [25], behavior of moving objects analysis [26] and game player behavior [27]. In addition, many studies have proposed to extend the framework of *FRIM* in several ways. We here give a brief review of the main related works.

Attempts to mine frequent-regular itemsets from incremental database/data streams based on the use of a tree-based structure, sliding window, and/or vertical data format are reported in [28,29]. To avoid difficulties in setting the support threshold, Amphawan [30] introduced the task of top-k frequent-regular itemset mining. A partition and estimation technique was proposed to increase the efficiency of this task [31]. Furthermore, a concise representation of top-k frequent-regular itemsets called top-k frequent-regular closed itemsets [32] has recently been proposed to avoid redundancy of discovered top-k frequent-regular itemsets. Focusing on rare itemsets, Kiran [33] and Surana [34] proposed to dynamically specify a maximum periodic threshold for each item and to use multiple support and regularity thresholds for mining rare-regular itemsets. Lastly, frequent-regular itemset mining was applied in elderly habit monitoring [35]. This can help extract regular activities that elders usually engage in when staying at home.

The previous approaches mentioned above usually consider only utility, frequency or regularity, which can provide only one aspect of information. Thus, in this paper, we introduce the discovery of a new kind of itemsets called high utility itemset with regular occurrence (*HUIR*). This task considers the utility value simultaneously with occurrence behavior (regular occurrence), which can increase the ability to provide a wider range of knowledge. Table 3 illustrates the characteristics of the itemsets mentioned above. There are six types of itemsets based on frequency, regularity and utility measures.

Table 3 Types of itemsets based on frequency, regularity and utility measures.

	Frequency of occurrence	**Regularity of occurrence**	**Utility of itemsets**
Frequent itemsets	×	-	-
Frequent-regular itemsets	×	×	-
High utility itemsets	-	-	×
Top-*k* high utility itemsets	-	-	×
High utility-frequent itesmets	×	-	×
High utility itemsets with regular occurrence[1]	-	×	×

[1] High utility itemsets with regular occurrence (*HUIR*) is our proposed method.

3 Problem Statement

In this section we describe the basic notations used for high utility itemset and frequent-regular itemset mining, including the concepts of the utility and regularity values of an itemset. Then, the problem of mining high utility itemsets with regular occurrence is introduced.

Let $I = \{i_1, i_2, \dots, i_n\}$ be a set of items. Each item $i_j \in I$ has it own external utility, denoted as $eu(i_j)$. The external utility of item i_j can be profit, cost, and other user-defined factors. A set $X \subseteq I$ is called k-itemset if X contains k items. A transactional database $D = \{t_1, t_2, \dots, t_m\}$ contains m transactions in which each transaction $t_p \in D$ is a 2-tuple containing: (i) a unique transaction identifier p (usually called TID for short); and (ii) an itemset $Y \subseteq I$ where each $i_j \in Y$ associates with internal utility, i.e. quantity of occurrence of i_j in transaction t_p, denoted as $iu(i_j, t_p)$. If $X \subseteq Y$, it can be said that X occurs in transaction t_p or transaction t_p contains X, denoted as p^X. Since X can occur several times in D, then the set of ordered TIDs of transactions containing X can be expressed as $TID^X = \{p^X, \dots, q^X\}$ where $p^X \leq q^X$ and $1 \leq p^X \leq m$.

3.1 Utility of Items/Itemsets

Definition 1. *The utility value of item i_j in transaction t_p is the profit, cost, or other user-defined factors of i_j in transaction t_p, defined as $u(i_j, t_p) = iu(i_j, t_p) \times eu(i_j)$.*

Definition 2. *The utility value of itemset X in transaction t_p is the summation of utilities of all items $i_j \in X$ that occur in transaction t_p, defined as $u(X, t_p) = \sum_{i_j \in X, X \in t_p} u(i_j, t_p)$.*

Definition 3. *The utility value of itemset X in database D is the summation of utilities of X that occur in transactions of D, defined as $u(X) = \sum_{X \in t_p, t_p \in D} u(X, t_p)$.*

Example 1. See Tables 1 and 2 for the external utility value of items and a transactional database containing 8 transactions with internal utilities. The utility of item 'a' can be calculated as $u(a) = u(a, t_1) + u(a, t_3) + u(a, t_5) + u(a, t_7) + u(a, t_9) = (3 \times 3) + (2 \times 3) + (2 \times 3) + (5 \times 3) + (3 \times 3) = 45$.

Based on the three definitions above, an itemset X is a high utility itemset if its utility $u(X)$ is not smaller than a user-specified utility threshold (σ_u). The problem of high utility itemset mining is the task of finding itemsets whose utility is not smaller than σ_u. The main challenge of this task is the large size of the search space due to the downward closure property [2], which cannot be held, i.e. a superset of a low utility itemset can be a high-utility one. Therefore, the concept of Transaction-Weighted Utility (*TWU*) [9], an over-estimated utility of an itemset that meets the downward closure property, is applied, which is defined as follows.

Definition 4. *The utility of transaction t_p is the summation of utility values of all items i_j that occur in transaction t_p, defined as* $u(t_p) = \sum_{i_j \in t_p} u(i_j, t_p)$.

Definition 5. *The transaction-weighted utility of itemset X is the summation of the utility of transactions that contain X, defined as* $TWU(X) = \sum_{X \in t_p, t_p \in D} u(t_p)$.

Example 2. From Table 1 and 2, the transaction-weighted utility of item 'a' can be computed as $TWU(a) = u(t_1) + u(t_3) + u(t_5) + u(t_7) + u(t_8) = 82 + 40 + 29 + 57 + 50 = 258$.

The value of $TWU(X)$ can be referred to as the over-estimated or the maximum/upper-bound utility value of itemset X and all supersets of X. Then, it can be said that an itemset X and all supersets of X are low utility itemsets if $TWU(X) < \sigma_u$. By this conclusion, the downward closure property can be held and we can apply the concept of TWU to eliminate low utility itemsets. However, TWU is a loose-over-estimated utility value. Therefore, Liu [9] proposed the new concept of tight over-estimated utility, which can estimate the utility of itemsets smaller than TWU. The following definitions are introduced for this purpose.

Definition 6. *Let $\succ$ be the order of items in set of items I. The remaining utility of X in the transaction t_p ordered by $\succ$ is the summation of the utility values of all items in t_p that were ordered after X, defined as* $ru(X, t_p) = \sum_{i_j \in t_p, i_j \succ X} u(i_j, t_p)$.

Definition 7. *The remaining utility of X in a database D is the summation of all remaining utility values of X in all transactions that contain X, defined as* $ru(X) = \sum_{X \in t_p, t_p \in D} ru(X, t_p)$.

Definition 8. *The tight over-estimated utility of X in database D is the summation between the utility and the remaining utility of X in database D, defined as* $tou(X) = u(X) + ru(X)$.

Example 3. For the external utility values and the transactional database in Tables 1 and 2, the tight over-estimated utility of itemset 'ad' can be $tou(ad) = u(ad) + ru(ad) = 189 + \big(ru(ad, t_1) + ru(ad, t_3) + ru(ad, t_7) + ru(ad, t_8)\big) = 189 + (5 + 16 + 10 + 5) = 225.$

Based on the notion of tight over-estimated utility it can be said that the merging of itemset X with any item i_j ordered after X is a low utility itemset if if $tou(X) < \sigma_{\mathrm{u}}$. Thus, we can apply the concept of tight over-estimated utility as a criterion to stop producing larger-size itemsets from itemset X, which can help cut down the search space.

3.2 Regularity of Occurrence

The concept of regularity is introduced to observe the occurrence behavior of itemsets in terms of regularity of occurrence. The regularity of an itemset is related to the gap of consecutive transactions in which the itemset does not occur in the database, which can be described as follows.

Definition 9. *The regularity of X before its first occurrence in transaction t_p is the gap of absence of X between the first transaction t_1 in the database and the first occurrence of X in transaction t_p, defined as $fr\ (X, t_p) = p$.*

Definition 10. *The regularity of X between two consecutive occurrences of X in transactions t_p and t_q is the gap of occurrence of X between t_p and t_q, defined as $r\big(X, t_p, t_q\big) = q - p$.*

Definition 11. *The regularity of X after its last occurrence in transaction t_z is the gap of absence from the last occurrence of X in t_z to the last transaction t_m of database, defined as $lr(X, t_z) = m - z$.*

Definition 12. *The regularity value of X in a database D is the maximal gap of absence based on the occurrence of X in database D, defined as $r(X) = max\big(fr(X, t_p)$, $(r\big(X, t_p, t_q\big)$, $r\big(X, t_q, t_w\big), \dots, r\big(X, t_y, t_z\big), lr(X, t_z))$.*

Example 4. For the transactional database in Table 2, the regularity of item 'a' can be calculated as $r(a) = max(lr(a, t_1), r(a, t_1, t_3), r(a, t_3, t_5), r(a, t_5, t_7)$, $r(a, t_7, t_8), lr(a, t_8)) = max(1, 2, 2, 2, 1, 1) = 2.$

With the regularity value of an itemset X (i.e. $r(X)$), we can say that X occurs at least once in every $r(X)$ consecutive transactions or X never disappears from the database more than $r(X)$ consecutive transactions.

Problem Statement. Given a database D, a utility threshold σ_u, and a regularity threshold σ_r, the task of mining high utility itemsets with regular occurrence is to find the complete set of itemsets: (i) whose utility values are not smaller than σ_u (i.e. itemsets giving profit $\geq \sigma_u$); and (ii) whose regularity values are not greater than σ_r (i.e. itemsets must occur at least once in σ_r consecutive transactions).

4 Proposed Method: New modified Utility List (NUL) and MHUIRA

In this section, we first describe the utility-list structure concept [7] used for maintaining utility and occurrence information of each item/itemset, after which the new modified utility list, called *NUL*, is introduced. Details of *MHUIRA* based on the use of *NUL* are described. Lastly, an example of *MHUIRA* with *NUL* is given.

4.1 Utility List and New Utility List Structure

As proposed by Liu [7], a utility list of an itemset X is an ordered set of 3-tuple entries used for maintaining utility and occurrence information of X, defined as

$$UL^X = \{< tid_1, u(X, t_{tid_1})_1, ru(X, t_{tid_1})_1 >, < tid_2, u(X, t_{tid_2})_2, ru(X, t_{tid_2})_2 >,$$

$$\ldots, < tid_m, u(X, t_{tid_m})_m, ru(X, t_{tid_m})_m >\}.$$

where each entry e of UL^X contains three pieces of information, i.e. (i) tid_e – a TID of transaction t_{tid_e} that contains X; (ii) $u(X, t_{tid_e})_e$– the utility value of X in transaction t_{tid_e}; and (iii) $ru(X, t_{tid_e})_e$ – the remaining utility of X in transaction t_{tid_e}, respectively.

Example 5. Let's consider item 'a' with external utility $eu(a) = 3$ (Table 1). Its occurrence in the database from Table 2 is $TID^a = \{1,3,5,7,8\}$. If the order of items is $a \prec b \prec c \prec \cdots \prec h$, then the utility list of 'a' can be expressed as $UL^a = \{< 1,9,73 >, < 3,6,34 >, < 5,6,19 >, < 7,15,42 >, < 8,9,41 >\}$. Each entry of UL^a – for example the first entry of UL^a is $< 1,9,73 >$ – lets us know that: (i) 'a' occurs in transaction t_1; (ii) the utility of 'a' in t_1 is 9; and (iii) the remaining utility of 'a' in t_1 (i.e. the summation of utility values of all items in t_1 ordered after 'a' is 73, respectively.

From the above, the utility list can maintain occurrence information simultaneously with utility and remaining utility, however, the calculation of utility 'XY' from the intersection of utility lists UL^X and UL^Y of itemsets X and

Y cannot produce a correct utility. For example, the utility value of $u(XY, t_x)$ in UL^{XY} is calculated from summation of $u(X, t_x)$ in UL^X and $u(Y, t_x)$ in UL^Y in which: (i) $u(X, t_x)$ is the summation of $u(X - i_j, t_x)$ and $u(X - i_k, t_x)$; and (ii) $u(Y, t_x)$ is the summation of $u(Y - i_v, t_x)$ and $u(Y - i_w, t_x)$. Then, in the case of itemsets X and Y having the same prefix (i.e. $X - i_j = Y - i_v$), the intersection produces $u(XY, t_x) = u(X - i_j, t_x) + u(X - i_k, t_x) + u(Y - i_v, t_x) + u(Y - i_w, t_x) = \left(2 \times u(X - i_j, t_x)\right) + u(i_j, t_x) + u(i_k, t_x)$, which is not correct. To alleviate this difficulty, Liu [7] proposed to repeatedly decrease each utility value $u(XY, t_x)$ in UL^{XY} by $u(X - i_j, t_x)$ of UL^{XY}. However, this consumes computational time.

From the above issue, we focus here on avoiding the repeated utility calculation process. Thus, the utility list is modified by adding a new piece of information, utility of prefix items, into each entry of the utility list, where each entry e of the new modified utility list (*NUL*) is in the form of $< tid_e, u(i, t_{tid_e})_e, ru(t_{tid_e})_e, up(t_{tid_e})_e >$ where $up(t_{tid_e})_e$ is the new additional information that expresses the utility value of prefix items.

Example 6. With the transactional database from Tables 1 and 2, itemset 'a' has $NUL^a = \{< 1{,}9{,}73{,}0 >, < 3{,}6{,}34{,}0 >, < 5{,}6{,}19{,}0 >, < 7{,}15{,}42{,}0 >, < 8{,}9{,}41{,}0 >\}$ and item 'b' has NUL^b as $\{< 2{,}10{,}0{,}0 >, < 5{,}6{,}13{,}0 >, < 7{,}2{,}40{,}0 >, < 8{,}2{,}39{,}0 >\}$, respectively. Since both items do not have a prefix, we can easily compute the utility of itemset 'ab' from the intersection of NUL^a and NUL^b. The itemset 'ab' first occurs in transaction t_5, then the third entry $e = < 5{,}6{,}26{,}0 >$ of NUL^a is merged with the second entry $f = < 5{,}6{,}13{,}0 >$ of NUL^b. The first entry g of NUL^{ab} is composed of: (i) $t_{tid_g} = 5$; (ii) $u(ab, t_5)_g = u(a, t_5)_e + u(a, t_5)_f - up(a, t_5)_e = 6 + 6 - 6 = 0$; (iii) $ru(ab, t_5)_g = ru(b, t_5)_f = 13$; and (iv) $up(ab, t_5)_g = u(a, t_5)_e = 6$, respectively. Thus, the entry g of NUL^{ab} is $< 5{,}12{,}13{,}6 >$. Also, this computation can be applied to other occurrences of itemsets 'ab'. Then, each entry of NUL^{ab} contains the correct utility of itemset 'ab'. With the use of NUL, we can avoid repeated utility calculation for each entry as in [7], which can also reduce computational time.

4.2 MHUIRA Algorithm

As mentioned above, *MHUIRA* not only applies the new utility list structure in order to efficiently maintain occurrence information and utility values but also employs the concept of remaining and overestimated utilities to cut down the search space. *MHUIRA* avoids repeatedly scanning the database by creating a simple list named t_{List} for storing the transaction utility of transactions in the database (used for computing the remaining utility of all items). *MHUIRA* consists of two steps: (i) *1-HUIRs identification* – the task of database scanning to capture occurrence information of each item into a simple 2D list named i_{List}, where each 1-dimension, e.g. 1_{List}, 2_{List}, ..., k_{List} of i_{List}, is used for maintaining 1-itemsets, 2-itemsets, …, k-itemsets, respectively; and (ii) *Mining HUIRs* – the process of mining a complete set of *HUIR*s from the itemsets contained in i_{List}.

1-HUIR identification. As detailed in Algorithm 1, t_{List} and 1_{List} of i_{List} are first created and initialized. Each transaction t_p is read (line 2-6) in order to: (i) compute transaction utility $u(t_p)$ of transaction t_p and then collect $u(t_p)$ from t_{List}; (ii) collect the occurrence information and utility value of each item i (occurring in transaction t_p) into its entries of 1_{List} (i.e. adding a new entry $e = < p, u(i, t_p), 0{,}0 >$ into NUL^i); (iii) compute and update the TWU of all items that appear in t_p; and (iv) compute the regularity value of each item i in t_p, respectively. Then, *MHUIRA* removes all irregular/low-utility items out of 1_{List} and the transaction utilities in t_{List} related to each occurrence of each irregular/low-utility items are updated (line 7-11). The remaining items in 1_{List} are then ordered by $\succ$. Next, the utility and remaining utility of each item i in 1_{List} are calculated (line 13-18). If the utility of item i is not smaller than the utility threshold, *MHUIRA* identifies item i as *1-HUIR* and then collects item i into *HUIR*. At the end of *1-HUIR* identification, we gain 1_{List} contained in i_{List} in which 1_{List} contains items that are potentially *HUIR*s used for generating longer *HUIR*s.

Algorithm 1. *1-HUIR* identification

Input. D : transactional database, σ_u: a utility threshold, σ_r: a regularity threshold
Output. i_{List}: a 2D list containing items with potentially to be *HUIR* in 1_{List}

1) create t_{List} , i_{List} and then create and initial 1_{List} in i_{List} for all items
2) **for each** transaction t_p in database D
3) compute and collect $u(t_p)$ of transaction t_p in t_{List}
4) **for each** item i in transaction t_p
5) compute $u(i, t_p)$ and then update NUL^i of i with $< p, u(i, t_p), 0{,}0 >$
6) compute TWU^i by $u(t_p)$ and regularity $r(i)$ of i by t_p
7) **for each** item i in 1_{List}
8) **if** $r(i) > \sigma_r$ or $TWU^i < \sigma_u$
9) **for each** entry $e = < tid_e, u(i, t_{tid_e})_e, ru(t_{tid_e})_e, up(t_{tid_e})_e >$ in NUL^i
10) decrease utility $u(t_{tid_e})$ of transaction t_{tid_e} in t_List by $u(i, t_{tid_e})_e$ of e
11) remove entry of i out of 1_{List}
12) sort 1_{List} based on the order of items $\succ$
13) **for each** item i in 1_{List}
14) initial value of $u(i)$ and $ru(i)$ to be 0

for each entry $e = < tid_e, u(i, t_{tid_e})_e, ru(\ t_{tid_e})_e, up(\ t_{tid_e})_e >$ in NUL^i
  decrease utility $u(t_{tid_e})$ of transaction t_{tid_e} in t_{List} by $u(i, t_{tid_e})_e$
  set $ru(\ t_{tid_e})_e$ to be $u(t_{tid_e})$ of t_{List}
  increase $u(i)$ by $u(i, t_{tid_e})_e$ and increase $ru(i)$ by $ru(\ t_{tid_e})_e$
if $u(i) \geq \sigma_{\mathrm{u}}$
  $HUIR = HUIR \cup i$

Mining HUIRs. As described in Algorithm 2, a simple list (2_{List}, used for storing 2-itemsets) is first created and initialized to be empty. Then, each item i in 1_{List} with $tou(i) \geq \sigma_{\mathrm{u}}$ is considered and merged with item j in 1_{List} located after item i in order to consider itemset ij. Both NUL^i and NUL^j of item i and j are then intersected in order to compute regularity $r(ij)$, utility $u(ij)$, remaining utility value $ru(ij)$, and to collect NUL^{ij} of itemset ij. The tight over-estimated utility of ij is then computed by utility, remaining utility and utility of prefix items (line 5). If regularity $r(ij)$ is not greater than the regularity threshol, a new entry of itemset ij with NUL^{ij} is created and inserted into 2_{List}.

Algorithm 2. Mining *HUIR*s

Input. i_{List}: a 2D-list of itemsets, σ_u: a utility threshold, σ_r: a regularity threshold
Output. $HUIR$: a complete set of *HUIR*s

1) **for each** item i in 1_{List} of i_{List} where $tou(i) \geq \sigma_{\mathrm{u}}$
2)   create 2_{List} in i_{List} and initial 2_{List} to be empty
3)   **for each** item j in 1_{List} of i_{List} (where $i \prec j$)
4)   intersect NUL^i and NUL^j of item i and j in order to calculate $r(ij)$, $u(ij)$, $ru(ij)$, $up(ij)$and to collect NUL^{ij} for further computation
5)   calculate $tou(ij)$ as $tou(ij) = u(ij) + ru(ij) - up(ij)$
6)   **if** $r(ij) \leq \sigma_{\mathrm{r}}$
7)   create an entry of itemset ij in 2_{List} with its $r(ij)$, $u(ij)$, $ru(ij)$, $up(ij)$ and NUL^{ij}
8)   **if** $u(ij) \geq \sigma_{\mathrm{u}}$
9) $HUIR = HUIR \cup ij$
10) **if** 2_{List} contains more than one itemsets
11) $GenLongerItemsets(2, i_{List})$

Procedure $GenLongerItemsets(k, i_{List})$
for each itemset u in k_{List} of $i_{List}$where $tou(ij) \geq \sigma_{\mathrm{u}}$
  create $(k+1)_{List}$ in i_{List} and initial $(k+1)_{List}$ to be empty
  for each itemset v in k_{List} of i_{List} (where $u \prec v$)
    intersect NUL^u and NUL^v of itemset u and v in order to calculate $r(uv)$, $u(uv)$, $ru(uv)$, $up(uv)$and to collect NUL^{uv} for further computation
    calculate $tou(uv)$ as $tou(uv) = u(uv) + ru(uv)$
    if $r(uv) \leq \sigma_{\mathrm{r}}$
      create an entry of itemset uv in $(k+1)_{List}$ with its $r(uv)$, $u(uv)$, $ru(uv)$, $up(uv)$ and NUL^{uv}
      if $u(uv) \geq \sigma_{\mathrm{u}}$

9) $HUIR = HUIR \cup uv$
10) **if** $(k+1)_{List}$ contains more than one itemset
11) $GenLongerItemsets(k+1, i_{List})$

Itemset ij is then collected in the set of *HUIR*s if the utility of ij is not smaller than the utility threshold. Next, *MHUIRA* repeats the same process of merging item i with another item j in 1_{List}. At the end of the merging of item i, 2_{List} contains 2-itemsets having: (i) item i as a prefix of itemset and (ii) regularity not greater than the regularity threshold, respectively. In addition, if 2_{List} contains more than one itemset, then 3_{List} is created and initialized. In a similar manner, each itemset X in 2_{List} is then merged with another itemset Y in 2_{List} in order to generate 3-itemset. This process is then continued until k_{List} is empty or contains only one itemset. *MHUIRA* repeatedly considers all items in a similar manner as item i. At the end of the mining step, we gain a set of *HUIRs* in which the regularities and utilities meet the thresholds.

4.3 Example of MHUIRA with NU

Let's consider a table of items' external utilities and a table of a transactional database with internal utilities (see Tables 1 and 2). Assume that the regularity and utility thresholds are set to be 3 and 30. The task of mining a complete set of HUIRs by MHUIRA with NUL is to find itemsets having regularity not greater than 3 and utility not smaller than 30, respectively. As shown in Fig. 1, MHUIRA first creates t_{List} with 8 entries for transaction $t_1 - t_8$ and i_{List} in which 1_{List} contains 8 entries for items $a, b, \ldots, h$. Next, each transaction is read. For the first transaction, $t_1 = \{a(3), c(8), d(2), e(1)\}$, its transaction utility is computed (i.e. $u(t_1) = (3 \times 3) + (8 \times 1) + (2 \times 30) + (1 \times 5) = 82$) and then added into the first entry of t_{List}. In addition, entries of item a, c, d, and e in 1_{List} are updated. Notice that each entry in 1_{List} contains item name, regularity, utility, remaining utility, and NUL, respectively. For the second transaction, $t_2 = \{b(5), f(3), g(5), h(20)\}$, the entry of transaction t_2 in t_{List} is updated with 339 and the entry of items b, f, g, and h in 1_{List} are updated. For the 3rd to the 8th transaction, MHUIRA performs in the same manner. Then, the entries of items g and h are eliminated from 1_{List}, since their regularity values are greater than $\sigma_r = 3$. Lastly, the remaining utility in each entry of NUL, total remaining utility, and total utility of each item are calculated (as shown in Figure 1).

Next, *HUIR* mining is performed. An item 'a' in 1_{List} is first considered and merged together with items b, c, d, e and f. For each merging, such as 'a' merged with 'b', $NUL^a = \{<1,9,73,0>, <3,6,34,0>, <5,6,19,0>, <7,15,42,0>, <8,9,41,0>\}$ and $NUL^b = \{<2,10,0,0>, <5,6,13,0>, <7,2,40,0>, <8,2,39,0>\}$ are intersected together in order to compute utility, tight over-estimated utility and regularity value of itemset 'ab' and to collect NUL^{ab} for further computation (*i.e.* $u(ab) = 12 + 17 + 11 = 40$, $tou(ab) = 40 + (13 + 40 + 39) = 132, r(ab) = 5$, and $NUL^{ab} = \{< 5,12,13,6 >, < 7, 17,40, 15 >, < 8,11,39,9 >\}$). Since regularity $r(ab)$ is greater than σ_r (itemset 'ab' does not regularly occur in the database), 'ab' should be removed out of consideration (based on the downward closure property of regularity [3]). Next, item 'a' is merged with item 'c' and NUL^a is intersected with $NUL^c = \{< 1,8,65,0 >, < 3,4,30,0 >, < 4,5,5,0 >, < 5, 1,12, \ 0 >, < 8,4, 35,0 >\}$ to compute $u(ac) = 17 + 10 + 7 + 13 = 47$, $tou(ac) = 47 + (65 + 30 + 12 + 35) = 189, r(ac) = 3$ and $NUL^{ac} = \{< 1, 17, 65,9 >, < 3,10,30, 6 >, < 5,7, 12,6 >, < 8,13,35,9 >\}$, respectively.

Scanning of t_1

t_{list}	t_1 : 82	t_2 : 0	t_3 : 0	t_4 : 0	t_5 : 0	t_6 : 0	t_7 : 0	t_8 : 0
1_{list}	a,1,9,0 <1,9,0,0>	b,0,0,0, -	c,1,8,0, <1,8,0,0>	d,1,60,0, <1,60,0,0>	e,1,5,0, <1,5,0,0>	f,0,0,0, -	g,0,0,0, -	h,0,0,0, -

Scanning of t_2

t_{list}	t_1 : 82	t_2 : 339	t_3 : 0	t_4 : 0	t_5 : 0	t_6 : 0	t_7 : 0	t_8 : 0
1_{list}	a,1,9,0 <1,9,0,0>	b,2,10,0 <2,10,0,0>	c,1,8,0, <1,8,0,0>	d,1,60,0, <1,60,0,0>	e,1,5,0, <1,5,0,0>	f,2,9,0 <2,9,0,0>	g,2,20,0 <2,20,0,0>	h,300,0,0, <2,300,0,0>

Scanning of t_1 - t_{10}

t_{list}	t_1 : 82	t_2 : 339	t_3 : 40	t_4 : 13	t_5 : 25	t_6 : 65	t_7 : 57	t_8 : 50
1_{list}	a,2,45,0 {<1,9,0,0>, <3,6,0,0>, <5,6,0,0>, <7,15,0,0>, <8,9,0,0>}	b,3,20,0 {<2,10,0,0>, <5,6,0,0>, <7,2,0,0>, <8,2,0,0>}	c,3,22,0 {<1,8,0,0>, <3,4,0,0>, <4,5,0,0>, <5,1,0,0>, <8,4,0,0>}	d,3,180,0, {<1,60,0,0>, <3,30,0,0>, <6,30,0,0>, <7,30,0,0>, <8,30,0,0>}	e,3,25,0, {<1,5,0,0>, <4,5,0,0>, <7,10,0,0>, <8,5,0,0>}	f,3,24,0 {<2,9,0,0>, <4,3,0,0>, <5,12,0,0>}	g,**4**,40,0 {<2,20,0,0>, <6,20,0,0>}	h,**4**,315,0 {<2,300,0,0>, <6,15,0,0>}

Calculation on utility and remaining utility

t_{list}	t_1 : 82	t_2 : 10	t_3 : 40	t_4 : 10	t_5 : 25	t_6 : 30	t_7 : 57	t_8 : 50
1_{list}	a,2,45,209, {<1,9,73,0>, <3,6,34,0>, <5,6,19,0>, <7,15,42,0>, <8,9,41,0>}	b,3,20,102 {<2,10,0,0>, <5,6,13,0>, <7,2,40,0>, <8,2,39,0>}	c,3,22,147, {<1,8,65,0>, <3,4,30,0>, <4,5,5,0>, <5,1,12,0>, <8,4,35,0>}	d,3,180,20, {<1,60,5,0>, <3,30,0,0>, <6,30,0,0>, <7,30,10,0>, <8,30,5,0>}	e,3,25,0 {<1,5,0,0>, <4,5,0,0>, <7,10,0,0>, <8,5,0,0>}	f,3,24,0 {<2,9,0,0>, <4,3,0,0>, <5,12,0,0>}		

Figure 1 Example of *1-HUIR* identification.

Since regularity $r(ac)$ is smaller than σ_r and $tou(ac)$ is greater than σ_u, it can be concluded that itemset 'ac' and its supersets are potentially *HUIR*s. Then, an entry for itemset 'ac' is created in 2_{List} along with its information. Also, due to utility $u(ac)$ being not smaller than σ_u, *MHUIRA* identifies 'ac' as an *HUIR*. The merging and intersection process continues for itemsets 'ad', 'ae', and 'af'. If at the end 2_{List} contains only 'ac', then *MHUIRA* stops considering item 'a' and all of its supersets, and changes its consideration to items 'b', 'c', 'd', 'e' and 'f' in the same manner as 'a'.

5 Experimental Evaluation and Complexity Analysis

In this section, we report our experimental studies to investigate the performance of *MHUIRA* with the new modified utility-list (*NUL*). To the best of our ability, we pushed the first effort to consider the regularity constraint together with the utility of the itemsets (as in [5]). Then, we only made a comparison between *MHUIRA with UL* (called *HURI-UL* in [5]) and *MHUIRA with NUL*. Moreover, we also made a modification of *HUI-Miner* [7] that scans the database twice, called *HUI-Miner-reg*, for mining high utility itemsets with regular occurrence in order to show the improvement of our proposed single-pass algorithm compared to the two-pass algorithm of *HUI-Miner*. In addition, time and space complexity analyses of our proposed method are provided. This can be the baseline for future approaches.

5.1 Experimental Setup

For the experiments in this paper, four datasets downloaded from [36] were used (as detailed in Table 4). The regularity and utility thresholds were set and varied from $1-30\%$ and $0.001-22\%$, respectively. The setting of the thresholds was based on the density of the data in each dataset. However, it was similar to previous approaches ([5,6,7,30,31,32]). *MHUIRA* with *UL* and *NUL* and *HUI-Miner-reg* were implemented in C and run on Xeon® 2.4 GHz with 64 GB of memory. Three kinds of experiments were conducted to observe runtime, memory consumption, and number of itemsets that are discovered. Experiments on runtime and memory usage were done based on two settings: (i) *a* fixed on highest value of regularity threshold and *a* varied on utility threshold; and (ii) *a* fixed on lowest value of utility threshold and *a* varied on regularity threshold. Meanwhile, the number of discovered *HUIR*s in each dataset was observed under the lowest utility and highest regularity thresholds.

5.2 Runtime

The runtimes of *MHUIRA with UL* and *NUL* under *a* variation of regularity and utility thresholds are depicted in Figures 2 and 3. In both figures, the runtime of *MHUIRA with UL* and *NUL* and *HUI-Miner-reg* increases as the regularity threshold increases (also for the decrease of the utility threshold). The reason is

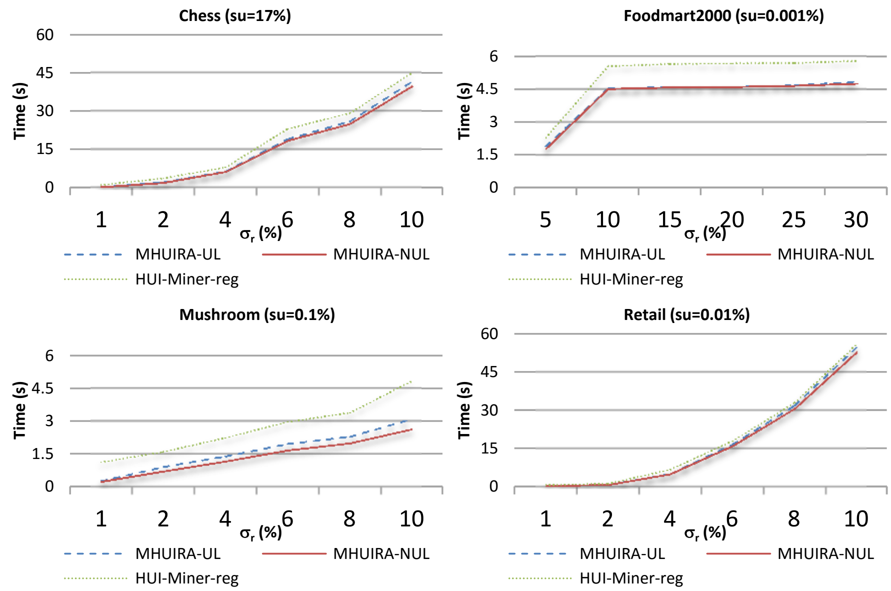


Figure 2 Runtime with variation of regularity threshold.

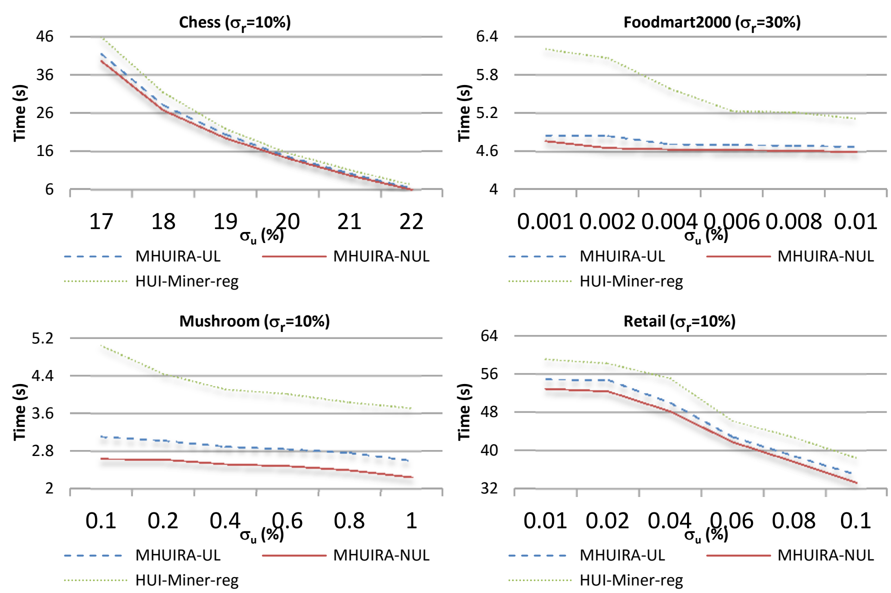


Figure 3 Runtime with variation of utility threshold.

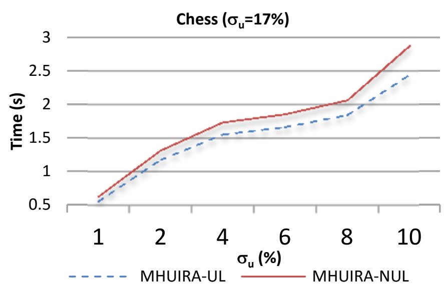
that with a higher regularity threshold (lower utility threshold), items/itemsets have more chance to meet the threshold. Then, the three algorithms have to spend more time to consider items/itemsets with a high regularity (low utility). In most cases, *MHUIRA with UL* and *NUL* are faster than *HUI-Miner-reg* since they can take advantage from avoiding repeatedly scanning the database. Moreover, *MHUIRA with NUL* is faster than *MHUIRA with UL*, since it can take advantage from *NUL*, which can help to avoid repeated calculation of utility. The percentage that runtime is faster is between 0 and 23%.

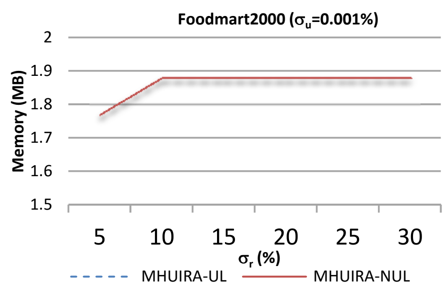
Table 4 Database characteristics.

Database	#items	Avg. transaction length	#transactions	type
Chess	75	37	3,196	dense
Foodmart2000	1,559	11	36,869	sparse
Mushroom	119	23	8,124	dense
Retail	16,469	10.3	88,162	sparse

5.3 Memory Usage

In Figures 4 and 5 the peak memory usage of the *MHUIRA with UL* and *NUL* with variation of the regularity and utility thresholds are illustrated. To do that for Figure 4, the regularity threshold was set to the highest value and the utility threshold was varied. On the other hand, the utility threshold was set to the lowest value of our consideration and the regularity threshold was varied (for Figure 5).





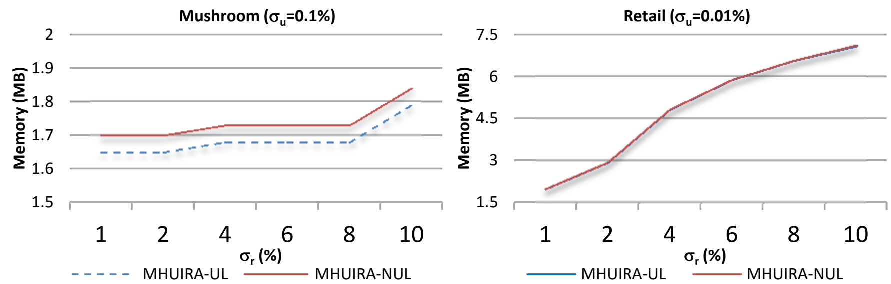


Figure 4 Memory usage with variation of regularity threshold.

With these lowest or highest thresholds, *MHUIRA* tends to produce a large amount of *HUIR*s, which lets us easily observe the highest memory usage of *MHUIRA*. From the figures it is obvious that in most cases the memory consumption of using *NUL* is higher than when using *UL*. This is because *NUL* stores additional information in each entry (i.e. the utility of the prefix itemset in a transaction). However, the amount of increase is between 0 and 15% of *UL*, which is a very small amount in megabytes and is not significant for computers in this era.

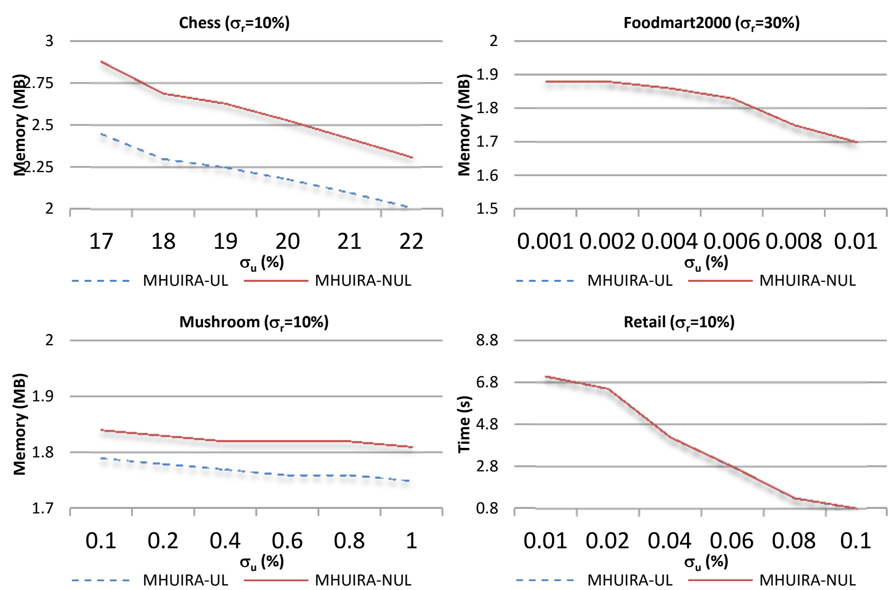


Figure 5 Memory usage with variation of utility threshold.

5.4 Number of Discovered Itemsets

To assess the potential of *MHUIRA* to discover *HUIRs*, experiments were conducted (in the same way as for the runtime investigation) to observe the number of discovered itemsets. Figure 6 shows the number of discovered itemsets with variation of the regularity threshold and the fixed value of the highest utility threshold. Meanwhile, Figure 7 indicates the number of discovered itemsets with a different variation. From both figures it can be seen that a high regularity threshold enables *MHUIRA* to generate more results than a low one. With a high regularity threshold there are thousands of itemsets that can meet the threshold. Meanwhile, a high utility threshold results in *MHUIRA* generating less itemsets than a low one (the reason is the contrast with the regularity threshold value).

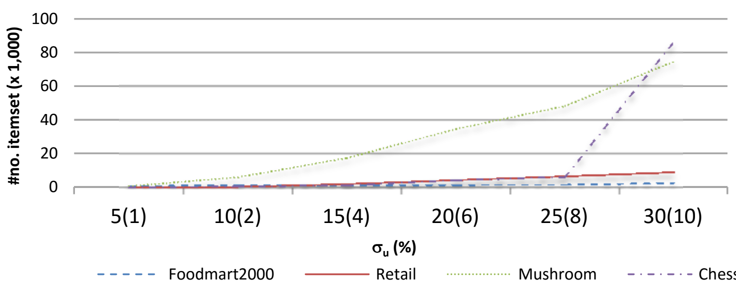


Figure 6 Number of *HUIRs* discovered by *MHUIRA* with variation of regularity threshold.

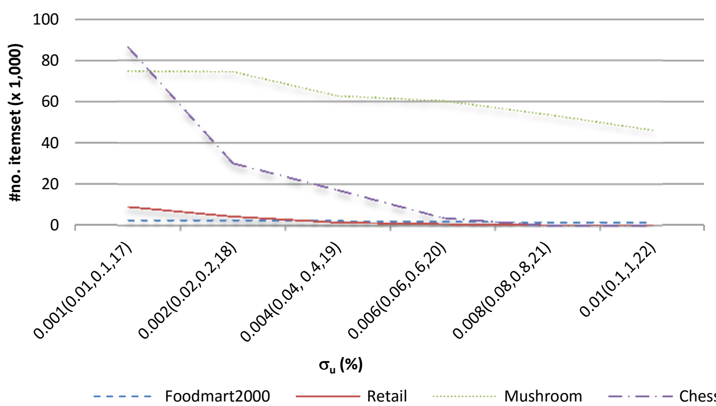


Figure 7 Number of *HUIRs* discovered by *MHUIRA* with variation of utility threshold.

5.5 Complexity Analysis

Lemma 1. Time complexity of *MHUIRA* is $O((nm) + (2^n m))$ where n is the number of items in set I and m is the number of transactions in database D.

Proof. As described in Algorithm 1, *MHUIRA* scans all transactions once, where each transaction may contain at most n items. Then, the time complexity of scanning the database is $O(nm)$. For mining all *HUIRs*, each item/itemset is merged with other items/itemsets having the same prefix. Then, the total number of itemsets to be regarded is 2^n. Also, each item/itemset may contain at most m entries of NUL. Thus, the number of intersections to calculate the utility value of itemsets (the main computation of *MHUIRA*) is $O(2^n m)$. Lastly, with the 2 main steps of *MHUIRA*, total computation of *MHUIRA* is equal to $O((nm) + (2^n m))$.

Lemma 2. Space complexity of *MHUIRA* is $O(n^2 m)$ of *NUL*'s entries, where n is the number of items in set I and m is the number of transactions in database D.

Proof. As described in Algorithm 1, 1_{List} is created for maintaining all single items with their corresponding *NUL*. Then, 1-List can contain at most nm entries of *NUL*. Also, for mining *HUIRs* with Algorithm 2, each item/itemset is considered and merged with previous items/itemsets (based on the ordered $\succ$ of items). Thus, each k_{List} is created to maintain k-itemsets with their corresponding utility list. If the maximum size of an itemset is n (i.e. the number of all single items), then *MHUIRA* creates n lists, where each list contains at most n itemsets with nm elements of the utility entry. Thus, the maximum space usage of *MHUIRA* is equal to $O(n^2 m)$ elements of the utility entry.

6 Conclusion

In this paper, we have proposed to add a regularity constraint to high utility itemset mining. Then, the problem of mining high utility itemsets with regular occurrence (*MHUIR*) was introduced. This can give information about "regular purchases by customers of high-profit products". To find such itemsets, an efficient single-pass algorithm named *MHUIRA* and a new modified utility-list structure (called *NUL*, used for maintaining utility and occurrence information) were presented. Experiments were conducted on both real and synthetic datasets and complexity analyses were also given to indicate the efficiency and effectiveness of *MHUIRA* and *NUL*. In the future, we will extend *MHUIR* by focusing on: (i) difficulties on assigning appropriate thresholds; (ii) redundancy of *HUIR*s; and (iii) finding *HUIRs* in different kinds of databases, respectively.

Acknowledgements

This work was financially supported by a research grant of Burapha University through the National Research Council of Thailand (Grant No. 47/2559).

References

[1] Agrawal, R., Imielinski, T. & Swami, A., *Mining Association Rules Between Sets of Items in Large Databases*, in Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, DC, USA, pp. 207-216, 1993.

[2] Agrawal, R. & Srikant, R., *Fast Algorithms for Mining Association Rules in Large Databases*, in Proceedings of the 1994 ACM SIGMOD international conference on Management of data, Minneapolis, MN, USA, pp. 487-499, 1994.

[3] Tanbeer, S.K., Ahmed, C.F., Jeong, B.S. & Lee, Y.K., *Discovering Periodic-Frequent Patterns in Transactional Databases*, in Proceedings of the 13th Pacific-Asia Knowledge Discovery and Data Mining conference (PAKDD 2009), Bangkok, Thailand, pp. 242-253, 2009.

[4] Chan, R., Yang, Q. & Shen, Y.D., *Mining High Utility Itemsets*, in Proceedings of IEEE International Conference on Data Mining (ICDM), Melbourne, Florida, USA, pp. 19-26, 2003.

[5] Amphawan, K. & Surarerks, A., *Pushing Regularity Constraint on High Utility Itemsets Mining*, in Proceedings of International Conference on Advanced Informatics: Concept, Theory and Application (ICAICTA 2015), Chonburi, Thailand, 2015.

[6] Liu, Y., Liao, W.K. & Choudhary, A., *A Two-Phase Algorithm for Fast Discovery of High Utility Itemsets*, in Advances in Knowledge Discovery and Data Mining, **3518**, pp. 689-695, 2005.

[7] Liu, M. & Qu, J., *Mining High Utility Itemsets without Candidate Generation*, in Proceedings of the 21st ACM International Conference on Information and Knowledge Management, Maui, HI, USA, pp. 55-64, 2012.

[8] Lin, C.W., Hong, T.P. & Lu, W.H., *An Effective Tree Structure for Mining High Utility Itemsets,* Expert Systems with Application, **38**(6), pp.7419-7424, 2011.

[9] Tseng, V.S., Wu, C.W., Shie, B.E. & Yu, P.S., *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, Washington DC, DC, USA, pp.18-27, 2010.

[10] Wu, C.W., Fournier-Viger, P., Yu, P.S. & Tseng, V.S, *Efficient Algorithms for Mining the Concise and Lossless Representation of Closed+ High Utility Itemsets*, in Proceedings of the 11th IEEE

International Conference on Data Mining, Vancouver, Canada, pp. 824-833, 2011.

[11] Ahmed, C., Tanbeer, S.K., Jeong, B.S. & Lee, Y.K., *Efficient Tree Structures for High Utility Pattern Mining in Incremental Databases*, IEEE TKDE, **21**(12), pp. 1708-1721, 2009.

[12] Lin, C-W., Hong, T.-P., Lan, G.-C., Wong, J.-W. & Lin, W.-Y., *Efficient Updating of Discovered High-Utility Itemsets for Transaction Deletion in Dynamic Databases*, Advanced Engineering Informatics, **29**(1), pp. 16-27, 2015.

[13] Lin, C-W., Gan, W. & Hong, T.P., *A Fast Updated Algorithm to Maintain the Discovered High-Utility Itemsets for Transaction Modification,* Advanced Engineering Informatics, **29**(3), pp. 562-574, 2015.

[14] Feng, L., Wang, L. & Jin, B., *UT-Tree: Efficient Mining of High Utility Itemsets from Data Streams*, Intelligence Data Analysis, pp. 585-602, 2013.

[15] Li, H.-F., Huang, H.-Y. & Lee, S.-Y., *Fast and Memory Efficient Mining of High-Utility Itemsets from Data Streams: With and Without Negative Item Profits*, Knowledge and Information Systems, **28**(3), pp. 495-522, 2011.

[16] Fournier-Viger, P., *FHN: Efficient Mining of High-Utility Itemsets with Negative Unit Profits*, in Proceedings of the 10th International Conference on Advanced Data Mining and Applications, Guilin, China, pp. 16-29, 2014.

[17] Wu, C.W., Shie, B-E., Tseng, V.S. & Yu, P.S., *Mining Top-k High Utility Itemsets*, in Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Beijing, China, pp. 78-86, 2012.

[18] Ryang, H. & Yun, U., *Top-k High Utility Pattern Mining with Effective Threshold Raising Strategies*, Knowledge-Based Systems, **76**, pp. 109-126, 2015.

[19] Tseng, V.S., Wu, C.-W., Fournier-Viger, P. & Yu, P.S., *Efficient Algorithms for Mining Top-K High Utility Itemsets*, IEEE Transactions on Knowledge and Data Engineering, **28**(1), pp. 54-67, 2015.

[20] Podpecan, V., Lavrac, N. & Kononenko, I,. *A Fast Algorithm for Mining Utility Frequent Itemsets*, in Proceedings of the International Workshop on Constraint-based Mining and Learning at ECML/PKDD, Warsaw, Poland, pp. 9-20, 2007.

[21] Sugunadevi, P. & Mythily, A.S., Efficient Algorithm for Mining High Utility Itemsets, The International Journal Of Science & Technology, **2**(5), pp. 250-253, 2014.

[22] Glynn, E.F, Chen, J. & Mushegain, A.R., *Detecting Periodic Patterns in Unevenly Spaced Gene Expression, Time Series Using Lombscargle Periodograms*, Bioinformatics, **22**(3), pp. 310-316, 2006.
[23] Luth, S., Herkel, J., Kanzler, S., Frenzel, C., Galle, P.R., Dienes, H.P., Schramm, C. & Lohse A.W., *Serologic Markers Compared with Liver Biopsy for Monitoring Disease Activity in Autoimmune Hepatitis*, Journal of Clinical Gastorenterology, **42**(8), pp. 926-930, 2008.
[24] Khallel, M., Dash, G., Choudhary, K. & Khan, M., *Medical Data Mining for Discovering Periodically Frequent Diseases From Transactional Databases*, Computational Intelligence in Data Mining, **31**, pp. 87-96, 2015.
[25] Engler, J., *Mining Periodic Patterns in Manufacturing Test Data*, in Proceedings of International Conference IEEE SoutheastCon 2008, Huntsville, Alabama, USA, pp. 389-395, 2008.
[26] Li, Z., Ding, B., Han, J., Kays, R. & Nye, P., *Mining Periodic Behaviors For Moving Objects*, in Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, Washington DC, DC, USA, pp. 1099-1108, 2010.
[27] Soulas, J. & Lenca, P., *Periodic Episode Discovery Over Event Streams,* in Proceedings of the 17th Portuguese Conference on Artificial Intelligence, EPIA 2015, Coimbra, Portugal, pp. 547-559, 2015.
[28] Tanbeer, S.K., Ahmed, C.F. & Jeong, B-S., *Mining Regular Patterns in Incremental Transactional Database*, in Proceedings of the 12th international Asia-Pacific web conference, Busan, Korea, pp. 375-377, 2010.
[29] Kumar, G. & Kumari, V., *Sliding Window Technique to Mine Regular Frequent Pattern in Data Streams Using Vertical Format*, in Proceedings of IEEE International Conference on Computational Intelligence Computing Research, Coimbatore, India, pp. 1-4, 2012.
[30] Amphawan, K., Lenca, P. & Surarerks, A., *Mining Top-K Periodic Frequent Patterns without Support Threshold*, in Proceedings of the 3rd International Conference on Advances in Information Technology, Bangkok, Thailand, pp. 18-29, 2009.
[31] Amphawan, K., Lenca, P. & Surarerks, A., *Mining Top-K Regular-Frequent Itemsets Using Database Partitioning and Support Estimation*, Expert Systems with Applications, **39**(2), pp. 1924-1936, 2012.
[32] Amphawan, K. & Lenca, P., *Mining Top-K Frequent-Regular Closed Patterns*, Expert Systems with Applications, **42**(21), pp. 7882-7894, 2015.
[33] Kiran, R.U. & Reddy, P.K., *Towards Efficient Mining of Periodic-Frequent Patterns in Transactional Databases*, in Proceedings of the

1st International Conference on Database and Expert Systems Applications, Bilbao, Spain, pp. 194-208, 2010.
[34] Surana, A., Kiran, R.U. & Reddy, P.K., *An Efficient Approach to Mine Periodic Frequent Patterns in Transactional Databases*, in Proceedings of International Workshops on New Frontiers in Applied Data Mining, Shenzhen, China, pp. 254-266, 2012.
[35] Amphawan, K., Soulas, J. & Lenca, P., *Mining Top-K Regular Episodes from Sensor Stream*s, in Proceedings of the 7th International Conference on Advances in Information Technology, Bangkok, Thailand, pp. 76-85, 2015.
[36] Fournier-Viger, P., *Spmf: An Open-Source Data Mining Library*, http://www.philippe-fournier-viger.com/spmf/, 2015.