

Efficient Discovery of High Utility-Frequent Itemsets using Cutoff Utility and Suffix Utility

Abstract. High Utility Itemset Model (HUIM) aims to discover all itemsets in transactional data that satisfy the user-specified *minimum utility* constraint. Two main limitations of this model are: (i) high computational cost (because the *utility* measure fails to satisfy the anti-monotonic property) and (ii) infrequent itemsets may be output as high utility itemsets. Most previous studies have tried to reduce the computational cost by searching the itemset lattice which is sorted with respect to items' Transaction Weighted Utility (*TWU*). It is because *TWU* is a convertible anti-monotonic measure, which represents the upper bound on the *utility* of itemsets. This paper argues that finding the desired itemsets from *TWU* ordered itemset lattice is inefficient because the depth-first search has to be performed on many items. This paper makes an effort to address these two limitations. A generic high utility-frequent itemset model is introduced to find all itemsets in the data that satisfy the user-specified *minimum support* and *minimum utility* constraints. Two new pruning measures, named *cutoff utility* and *suffix utility*, are introduced to reduce the computational cost of finding the desired itemsets. A single phase fast algorithm, called High Utility Frequent Itemset Miner (HU-FIMi), is introduced to discover the itemsets efficiently. Experimental results demonstrate that the proposed algorithm is efficient.

Keywords: Data mining · Itemset mining · Utility itemset

1 Introduction

The High Utility Itemset Model (HUIM) extends Frequent Itemset Model (FIM) by taking into account information regarding the *frequency* of an item in a transaction (called internal utility) and the importance of an item in the data (called external utility). Given a transactional database and a utility database, the aim of HUIM is to discover all interesting itemsets whose *utility* in the database is no less than the user-specified *minimum utility* (*minUtil*) constraint. The utility of an *itemset* is the summation of its utilities in all the transactions. The classic application of HUIM is market-basket analysis. It consists of analyzing which sets of items purchased by customers generate a sufficient revenue for a retailer. An example of utility itemset generated in the Yahoo! Japan retail data¹ is:

$$\{Nintendo3Ds_game, Playstation4_game\} \quad [utility = 604,231 \text{ ¥}].$$

The above utility itemset says that the *income* (or *utility*) generated from the simultaneous purchases of 'nintendo3Ds_game' and 'playstation4_game' is 604,231

¹ More details of this dataset are presented in latter parts of this paper

¥. This information may be useful to the retailer, because it disproves the general assumption of expecting little revenue from the customers purchasing games for competing products simultaneously. HUIM has many other applications, such as website click stream analysis, cross-marketing and bio-medical applications [2].

The popular adoption and successful industrial application of HUIM has been hindered by the following two obstacles:

1. **High computational cost of the model:** The itemsets generated by *utility* measure do not satisfy the convertible anti-monotone, convertible monotone, or convertible succinct properties [6]. To reduce the search space in the itemset lattice, most algorithms [2] find secondary items by pruning all items that have *local utility*² less than *minUtil*. The *local utility* is a convertible anti-monotonic measure, which represents an upper bound on the *utility* of itemsets. Next, primary items are generated by ordering the secondary items in ascending order of their *local utility*. Finally, all high utility itemsets are generated by recursively mining the projected databases of each primary item. Since the creation of projected databases requires a database scan, the computational cost of an high utility itemset mining algorithm depends primarily on the size of primary items. We have observed that finding primary items based on *local utility* order is inefficient (or computationally expensive), because such an order often generate a large number of primary items, thereby increasing the number of database scans.
2. **Infrequent itemsets may be generated as the utility itemsets:** Since HUIM determines the interestingness of an itemset without taking into account its *support* within the data, uninteresting itemsets with very low *support* may be generated as the utility itemsets. In our empirical study, we have observed a significant portion of high utility itemsets generated by HUIM have appeared very few times in the entire data. It is because the *utility* measure is sensitive to items with high external utility values. Additionally, directly pushing minimum support constraint into existing HUIM algorithms [2] is not an efficient solution to this problem. It is because such an algorithm cannot exploit the relationship between the *support* and *utility* measures to reduce the search space effectively.

This paper makes an effort to address these two problems.

We propose a generic High Utility-Frequent Itemset Model (HU-FIM) to find all high utility-frequent itemsets in the data that satisfy the user-specified *minimum support* (*minSup*) and *minUtil* values. The usage of *minSup* facilitates us to prune uninteresting itemsets having low *support* in the data. The itemsets generated by the proposed model do not satisfy the convertible anti-monotonic property, convertible monotonic property, or convertible succinct property. Two new pruning measures, “*cutoff utility*” (*CU*) and “*suffix utility*” (*SU*), have been introduced to reduce the search space and the computational cost of HU-FIM. The *CU* measure tries to reduce the search space by exploiting the relationship

² Since the *local utility* measure generalizes the *TWU* measure by taking into account itemsets, we use the former measure throughout this paper for brevity

between the *minSup* and external utility of an item. It states that if the *utility* of an item is less than the product of its *external utility* and *minSup*, then neither the item nor its supersets will be high utility-frequent itemsets. Given a **list of items in utility descending order**, the *SU* of an item represents the sum of utilities of remaining items. This measure states that if the sum of *utility* and *SU* of an item in the *utility* ordered list is less than *minUtil*, then the mining algorithm can be terminated as no more high utility-frequent itemsets will be generated from the data. Thus, *SU* can be used to reduce the search space of the model effectively. A single phase algorithm, called High Utility-Frequent Itemset Miner (HU-FIMi), has been proposed to find high utility-frequent itemsets efficiently. HU-FIMi is based on EFIM [10]. Experimental results demonstrate that the proposed model can discover useful information, and HU-FIMi is efficient.

The rest of the paper is organized as follows. Related work is presented in Section 2. Section 3 introduces HU-FIM. Section 4 briefly describes EFIM. Section 4 introduces the proposed algorithm. Experimental results are reported in Section 5. Section 6 concludes the paper with future research directions.

2 Related Work

FIM is an important model in data mining. The main limitation of this model is that it ignores the crucial information regarding the importance of items and their occurrence *frequency* in every transaction. Yao et al. [8] introduced HUIM by taking into account the importance of items and their occurrence *frequency* in every transaction. To circumvent the fact that the utility is not anti-monotonic and to find all high utility itemsets, several HUIM algorithms (e.g. Two-Phase [4] and UP-Growth+ [7]) have employed *local utility* to reduce the search space.

Recently, single phase algorithms (e.g. d2HUP [3] and EFIM [10]) to mine high utility itemsets were developed to avoid the problem of candidate generation. These algorithms use upper bounds that are tighter than the *local utility* to prune the search space and can immediately obtain the exact utility of any itemset to decide if it should be output. Zhang et al. [9] conducted an empirical study on various HUIM algorithms and concluded that EFIM has consistently shown better performance over other algorithms. In this paper, we push the proposed pruning measures into EFIM to find desired itemsets efficiently.

(Fazla and Zhang [5] discussed high utility itemset mining using multiple minimum supports and multiple minimum utility values. Due to incorrectness of their work, we are not discussing it in our paper though their work is closed related to our work. More details are presented in supplementary material)

3 Proposed Model: High Utility-Frequent Itemset

Let $I = \{i_1, i_2, \dots, i_m\}$, $m \geq 1$, be a set of items. Each item $i_j \in I$ is associated with a positive number $p(i_j)$ known as **external utility**. The external utility of an item represents its relative importance to the user. The **utility database**, UD , is a set of all items in I and their respective external utility values. That is,

$UD = \{(i_1, p(i_1)), (i_2, p(i_2)), \dots, (i_m, p(i_m))\}$. A **transactional database** is a set of transactions $D = \{T_1, T_2, \dots, T_n\}$ such that for each $T_c \in D$, $T_c \subseteq I$ and T_c has a unique identifier $c \in \mathbb{Z}^+$ called its transaction-identifier (or *tid*). Every item $i_j \in T_c$ has a positive number $q(i_j, T_c)$, called its **internal utility**. The internal utility of an item generally represents its *frequency* in a transaction.

Table 1. Market-basket database

<i>tid</i>	Items	<i>tid</i>	Items
1	(a, 2), (b, 3), (f, 2)	5	(a, 1), (b, 2), (c, 1)
2	(a, 2), (c, 1), (d, 3), (e, 2)		(d, 4), (g, 2)
3	(a, 3), (b, 1), (h, 2)	6	(c, 3), (d, 2), (f, 3),
4	(c, 2), (d, 3), (e, 1)		(e, 1)
		7	(b, 3), (d, 4)

Table 2. Price of items

Item	price	Item	price
a	200	e	200
b	300	f	500
c	200	g	200
d	400	h	300

Example 1. Consider the market-basket (or transactional) database shown in Table 1. The set of all items in the database, i.e. $I = \{a, b, c, d, e, f, g, h\}$. The *prices* (or external utilities) for all items in Table 1 are shown in Table 2. Let the unit for these *prices* be Japanese Yen. The first transaction in Table 1 indicates that a customer has purchased 2 quantities of item *a*, 3 quantities of item *b*, and 2 quantities of item *f*. These quantities represent the internal utilities of the items appearing in the first transaction.

Definition 1. (Utility of an item in a transaction) The utility of an item i_j in a transaction T_c , denoted as $u(i_j, T_c)$, represents the product of its internal and external utility values. That is, $u(i_j, T_c) = p(i_j) \times q(i_j, T_c)$.

Example 2. Continuing with the previous example, the *utility* (or *income*) of an item *a* in the first transaction, i.e., $u(a, T_1) = p(a) \times q(a, T_1) = 2 \times 200 = 400$ ¥.

Definition 2. (Utility of an itemset in a transaction) Let $X \subseteq I$ be an itemset. An itemset is a *k*-itemset if it contains *k* items. The utility of an itemset X in a transaction T_c , denoted as $u(X, T_c) = \sum_{i_j \in X} u(i_j, T_c)$ if $X \subseteq T_c$.

Example 3. The set of items ‘*a*’ and ‘*b*’, i.e., $\{a, b\}$ (or ‘*ab*’ in short) is an itemset. This is a 2-itemset. The utility (or *income*) of *ab* in T_1 , $u(ab, T_1) = u(a, T_1) + u(b, T_1) = 400 + 900 = 1300$ ¥.

Definition 3. (Utility of an itemset in database) The utility of an itemset X in the database D , denoted as $u(X) = \sum_{T_c \in g(X)} u(X, T_c)$, where $g(X)$ is the set of transactions containing X .

Example 4. In Table 1, *ab* has appeared in T_1, T_3 and T_5 . Therefore, $g(x) = \{T_1, T_3, T_5\}$. The *utility* (or *income*) of *ab* in each of these transactions, i.e., $u(ab, T_1) = 1300$ ¥, $u(ab, T_3) = 900$ ¥ and $u(ab, T_5) = 800$ ¥. Therefore, the utility (or *income*) of *ab* in whole data, i.e., $u(ab) = 1300 + 900 + 800 = 3000$ ¥.

Definition 4. (Frequent itemset) The support of an itemset X in D , denoted as $s(X) = |g(X)|$, where $|g(X)|$ represents the total number of transactions containing X in D . An itemset X is said to be **frequent** if $s(X) \geq \text{minSup}$, where *minSup* represents the user-specified minimum support.

Example 5. The support of ab in Table 1, i.e., $s(ab) = |g(ab)| = |\{T_1, T_3, T_5\}| = 3$. If $minSup = 3$, then ab is a frequent itemset because $s(ab) \geq minSup$.

Definition 5. (High utility-frequent itemset) A frequent itemset X is a high utility-frequent itemset if its $u(X) \geq minUtil$, where $minUtil$ represents the user-specified minimum utility value. An utility-frequent itemset X is expressed as X [support = $s(X)$, utility = $u(X)$].

Example 6. If the user-specified $minUtil = 2600$, then the frequent itemset ab is a high utility-frequent itemset because $u(ab) \geq minUtil$. This itemset is expressed as ab [support = 3, utility = 3,000 ¥].

Definition 6. (Problem definition) Given a transactional database D and a utility database UD , the problem of HU-FIM is to find all itemsets in D that have support $\geq minSup$ and utility $\geq minUtil$. The support of an itemset can be expressed in percentage of $|D|$. It is interesting to note that HUIM is a special case of the problem HU-FIM when $minSup = 0$.

4 EFIM and its limitations

Philippe et.al [10] introduced EFIM to find high utility itemsets. Since the proposed algorithm extends EFIM to find high utility-frequent itemsets, we briefly describe the steps of the latter algorithm. First, EFIM scans the database and calculates the *local utility* (see Definition 9) for all items. Next, secondary items are generated by pruning all items that have *local utility* less than $minUtil$. These **secondary items are later sorted in local utility ascending order**. Let \succ denote this sorted list of secondary items. Next, the *sub-tree utility* (see Definitions 10) for all secondary items is determined by performing another scan on the database. Next, items with *sub-tree utility* more than $minUtil$ as considered as primary items. Finally, for each primary item, a projected database is constructed and recursively mined to find all high utility itemsets.

Definition 7. (Items that can extend an itemset). Let α be an itemset. Let $E(\alpha)$ denote the set of all items that can be used to extend α according to the depth-first search, that is $E(\alpha) = \{z, z \in I \wedge z \succ \alpha, \forall \alpha \in \alpha\}$.

Example 7. Consider the database from Table 1. Let \succ be alphabetical order and $\alpha = \{a\}$, Then $E(\alpha) = \{b, c, d, e, f, g, h\}$.

Definition 8. (Remaining utility). The remaining utility of X in a transaction T_c is defined as $re(X, T_c) = \sum_{i \in T_c \wedge i \succ x \forall x \in X} u(i, T_c)$.

Example 8. The remaining utility of ac in T_5 , $re(ac, T_5) = 4 \times 400 + 2 \times 200 = 2000$.

Definition 9. (local utility). Let α an itemset and item $z \in E(\alpha)$. The LocalUtility of z with respect to α is $lu(\alpha, z) = \sum_{T \in g(\alpha, \{z\})} [u(\alpha, T) + re(\alpha, T)]$.

Definition 10. (Sub-tree utility) Let be an itemset α and an item z that can extend α according to the depth search ($z \in E(\alpha)$). The Sub-tree utility of z with respect to α is $stu(\alpha, z) = \sum_{T \in g(\alpha \cup \{z\})} [u(\alpha, T) + u(z, T) + \sum_{i \in T \wedge i \in E(\alpha \cup \{z\})} u(i, T)]$.

Example 9. Continuing with the previous example, let $\alpha = a$ and c be an item that can extend α in depth-first. The *sub-tree utility* of c with respect to α , i.e., $stu(\alpha, c) = (((2 \times 200) + (1 \times 200) + (3 \times 400 + 2 \times 200)) + ((1 \times 200) + (1 \times 200) + (4 \times 400 + 2 \times 200))) = 4600$.

The limitations of EFIM are as follows (i) EFIM finds high utility itemsets without taking into account their *support* information. As a result, the itemsets with low frequency can be generated as high utility itemsets. (ii) Since the creation of projected database for a primary item requires a scan on the data, the computational cost of EFIM mostly depends on the size of primary items. We have observed that EFIM is computationally expensive (or impractical on very large databases). It is because the *local utility* order of items generates too many primary items, thus increases the number of database scans.

5 Proposed Algorithm

HU-FIMi is a single phase algorithm that extends EFIM [10] to find high utility-frequent itemsets in a transactional database. To achieve better performance over EFIM, HU-FIMi exploits different ordering of items and introduces two new pruning measures to reduce the computational cost of finding the desired itemsets. The algorithm HU-FIMi is presented in Algorithms 1 and 2. The algorithm has the following steps: (i) finding secondary items, i.e., items whose supersets can be high utility-frequent itemsets, (ii) finding candidate items from secondary items and (iii) finding primary items from candidate items, and (iv) finding all high utility-frequent itemsets by recursively mining all primary items. We briefly explain each of these steps along with the pruning measures.

5.1 Finding secondary items

Since the *utility* measure is neither monotonic nor anti-monotonic function, the high utility-frequent itemsets generated by the proposed model do not satisfy the anti-monotonic property. To reduce the search space, we initially find secondary items constituting of all items whose supersets may generate high utility-frequent itemsets. The secondary items (see Definition 12) are identified by calculating each items' *local utility* (see Definition 9), *cutoff utility* (see Definition 11) and *support*. Since *cutoff utility* is a new pruning measure, we define this measure.

Definition 11. The cutoff utility of an item i_j , denoted as $cu(i_j)$, is the product of its external utility and $minSup$. That is, $cu(i_j) = p(i_j) \times minSup$.

Example 10. Consider the item 'g' in Table 1. The external utility of 'g,' i.e., $p(g) = 200$. If the user-specified $minSup = 3$, then 'g' should appear at least in three transactions with internal utility of 1. Therefore, the cutoff utility 'g' must have to be a high utility-frequent itemset is 600 ($= p(g) \times minSup$).

Algorithm 1 HU-FIMi

-
- 1: **input** : D : a transaction database, $minutil$: a user-specified threshold, $minSup$: a user-specified threshold
 - 2: **output** : the set of high utility-frequent itemsets
 - 3: Let α denote an itemset that needs to be extended. Initially, set $\alpha = \phi$;
 - 4: Scan the database D to determine the TWU , $support$ and $utility$ for every item $i_j \in I$. bin-arrays [10] can be used to calculate TWU , $support$ and $utility$ of the items effectively.
 - 5: $Secondary(\alpha) = \{i | i \in I \wedge lu(\alpha, i) \geq minutil \wedge s(i) \geq minSup \wedge u(i) \geq cu(i)\}$;
 - 6: Let \succ be the total order of $utility$ descending values on $Secondary(\alpha)$;
 - 7: Scan D to remove each item $i \notin Secondary(\alpha)$ from the transactions, sort items in each transaction according to \succ , and delete empty transactions;
 - 8: Sort transactions in D according to \succ_T ;
 - 9: Calculate $suffix\ utility$ for each item $i \in Secondary(\alpha)$.
 - 10: Let $pi(\beta)$ denote the set of all items in $Secondary(\alpha)$ that have $u(i) + su(i) \geq minUtil$;
 - 11: Calculate sub-tree utility for all items in $pi(\beta)$ by scanning database D , once using utility-bin array;
 - 12: $Primary(\alpha) = \{z \in pi(\beta) | stu(\alpha, z) \geq minUtil\}$;
 - 13: $RecursiveSearch(\alpha, D, Primary(\alpha), Secondary(\alpha), minUtil, minSup)$;
-

The pruning of items using *cutoff utility* is given in Property 1.

Property 1. For an item i_j , if $u(i_j) < cu(i_j)$, then neither i_j nor its supersets can be high utility-frequent itemsets.

Example 11. The *utility* of g in Table 1, $u(g) = 400$. Since $u(g) < cu(g)$, ‘ g ’ and its supersets cannot be utility-frequent itemsets.

Definition 12. (Secondary item) An item $i_j \in I$ is a secondary item if $lu(i_j) \geq minUtil$, $u(i_j) \geq cu(i_j)$ and $s(i_j) \geq minSup$.

Example 12. The secondary items in Table 1 are a, b, c, d and e . The items f, g and h are not secondary items because $s(f) < minSup$, $u(g) < cu(g)$ and $lu(h) < minUtil$, respectively.

5.2 Finding candidate items

The secondary items generated in the above step constitute of both high utility-frequent items and uninteresting items. To reduce the computational cost of finding the desired itemsets, we need to identify those secondary items whose depth-first search in the itemset lattice (or projected databases) will result in finding all high utility-frequent itemsets. To find such items, we introduce new pruning measure, called *suffix utility* (see Definition 13), by exploiting items’ *utility* descending order. The *suffix utility* facilitates us to define a **novel terminating condition**, which does not exist in any of the previous utility itemset mining algorithms. That is, if the sum of *utility* and *suffix utility* of an item is

Algorithm 2 RecursiveSearch

```

1: input :  $\alpha$ : an itemset,  $\alpha - D$ ; the  $\alpha$  projected database,  $Primary(\alpha)$ : the primary
   items of  $\alpha$ ,  $Secondary(\alpha)$ : the secondary items of  $\alpha$ , the  $minUtil$  threshold, the
    $minSup$  threshold
2: output: the set of high utility-frequent itemsets that are extensions of  $\alpha$ 
3: for each  $item_i \in Primary(\alpha)$  do
4:    $\beta = \alpha \cup \{i\}$ ;
5:   Scan  $\alpha$ -D to calculate  $u(\beta)$ ,  $s(\beta)$  and create  $\beta$ -D; //uses transaction merging
   from EFIM
6:   if  $u(\beta) \geq minUtil$  &&  $s(\beta) \geq minSup$  then
7:     output  $\beta$ 
8:   end if
9:   if  $s(\beta) \geq minSup$  then
10:    Calculate  $stu(\beta, z)$  and  $lu(\beta, z)$  for all item  $z \in Secondary(\alpha)$  by scanning
     $\beta$ -D once, using two utility-bin array;
11:     $Primary(\beta) = \{z \in Secondary(\alpha) | stu(\beta, z) \geq minUtil\}$ ;
12:     $Secondary(\beta) = \{z \in Secondary(\alpha) | lu(\beta, z) \geq minUtil\}$ ;
13:    Search( $\beta$ ,  $\beta$ -D,  $Primary(\beta)$ ,  $Secondary(\alpha)$ ,  $minUtil$ ,  $minSup$ );
14:   end if
15: end for

```

less than $minUtil$, then mining process can be terminated as no further desired itemsets will be generated (see Lemma 1).

The *suffix utility* of an item i_j , $su(i_j) = u(i_{j+1}) + su(i_{j+2})$. If i_{j+2} represents the last time in the sorted list, then $su(i_{j+2}) = 0$. Thus, the time complexity of *suffix utility* measure is $O(1)$. Definition 14 defines **candidate items** generated using both *utility* and *suffix utility* measures.

Property 2. The *utility* of an itemset X will always be less than or equal to sum of *utility* of its items. That is, $u(X) \leq \sum_{i_j \in X} u(i_j)$.

Definition 13. (Suffix utility) Let $S = \{i_1, i_2, \dots, i_k\} \subseteq I$ be an ordered list of secondary items such that $u(i_1) \geq u(i_2) \geq \dots \geq u(i_k)$. The suffix utility of an item $i_j \in S$, denoted as $su(i_j)$, is sum of utilities of remaining items in the list. That is, $su(i_j) = \sum_{p=j+1}^{|S|} u(i_p)$. For the last item in S , $su(i_k) = 0$.

Example 13. The secondary items in *utility* descending order are d, b, a, c and e . The *suffix utility* of d , $su(d) = u(b) + u(a) + u(c) + u(e) = 6500 \text{ ¥}$.

Property 3. If $u(i_j) + su(i_j) < minUtil$, $i_j \in S$, then neither i_j nor the supersets generated from its projected database will be high utility-frequent itemsets.

Property 4. The suffix utility is a monotonically decreasing function. That is, the $su(i_j) \geq su(i_k)$, where $j < k$ and $i_j, i_k \in S$.

Lemma 1. Let $S = \{i_1, i_2, \dots, i_k\} \subseteq I$ be an ordered list of secondary items in *utility* descending order, i.e., $u(i_1) \geq u(i_2) \geq \dots \geq u(i_k)$. For an item $i_j \in S$, if $u(i_j) + su(i_j) < minUtil$, then no more highly utility-frequent itemsets will be generated from the projected databases of the remaining items in S .

Proof. According to Property 3, if $u(i_j) + su(i_j) < minUtil$, then no more highly utility-frequent itemsets will be generated from the projected database of i_j . Now let us consider another item i_k , $1 \leq j < k \leq |S|$. Since S is in *utility* descending order, $u(i_k) \leq u(i_j)$ and $su(i_k) \leq su(i_j)$ (see Property 4). Thus, $u(i_k) + su(i_k) \leq u(i_j) + su(i_j) < minUtil$. Thus, neither i_k nor the itemsets generated from its projected database will be high utility-frequent itemsets. Hence proved.

Definition 14. (*candidate item*) A secondary item $i_j \in S$ is a candidate item if $u(i_j) + su(i_j) \geq minUtil$.

5.3 Finding primary items

In majority cases, the candidate items generated in the previous step form the primary items. However, in few cases, especially when mining itemsets at low $minUtil$ values, suffix utility is inadequate. It is because most secondary items will become candidate items. To handle such cases, we generate primary items by calculating the *sub-tree utility* for candidate items. The calculation of *sub-tree utility* is a computational expensive step because it needs a complete database scan. We recommend eliminating this step when finding high utility-frequent itemsets at high $minUtil$ values. In this paper, we are providing this step for completeness. **(Please note that the primary items generated by EFIM and HI-FIMi algorithms can be different as both algorithms utilize different ordering of secondary items.)**

Definition 15. (*Primary item*). Let C denote the set of candidate item. A candidate item $i_j \in C$ is a primary item if $stu(i_j) \geq minUtil$.

5.4 Recursive mining of primary items

For each primary item, construct its projected database, and recursively mine the projected database until the projected database is empty.

6 Experimental Results

In this section, we first show that HU-FIMi performs better than EFIM. Later we evaluate only the performance of HU-FIMi as there exists no algorithm to find high utility-frequent itemsets. Both algorithms were written in C++ and executed on a machine with 1.5GHz processor and 4GB RAM. The experiments have been conducted using synthetic (**T10I4D100K** and **Retail**) and real-world (**Yahoo**) databases. The **T10I4D100K** database was generated using SPMF toolkit [1]. This database contains 870 items and 100,000 transactions. The minimum and maximum transaction lengths of this database are 1 and 29, respectively. The **Retail** database was also provided by SPMF toolkit. The internal and external utilities of the items are synthetically generated by SPMF toolkit. This database contains 16,470 items and 88,162 transactions. The minimum and maximum transaction lengths are 1 and 76, respectively. The **Yahoo** database

represents a portion of retail data generated by Yahoo! Japan. It contains 7,290 items and 93,113 transactions. The minimum and maximum transaction lengths are 1 and 24, respectively.

To evaluate EFIM and HU-FIMi algorithms, we find high utility-frequent itemsets by setting $minSup = 0$. Due to page limitation, we present the experimental results only for Yahoo database. Other experimental results were provided in supplementary material. Figure 1(a) show the number of primary items generated by EFIM and HU-FIMi algorithms. It can be observed that HU-FIMi has generated less number of primary items for any given $minUtil$ value (less is preferable as each primary item requires a database scan). Figure 1(b) show the number of nodes explored in the itemset lattice to find desired itemsets. It can be observed that proposed algorithm has explored relatively less number of nodes as compared against EFIM. Figure 1(c) shows the total runtime of EFIM and HU-FIMi algorithms to find all high utility itemsets. It can be observed that HU-FIMi has taken less runtime over EFIM. The performance improvement of HU-FIMi over EFIM is mainly due to the *suffix utility*, which facilitated us to find all high utility itemsets with relatively few primary items. Figure 1(d) shows the memory consumed by EFIM and HU-FIMi in Yahoo database. It can be observed HU-FIMi has consumed slightly more memory than EFIM. It is because HU-FIMi has to additionally store the *support* information of an itemset.

Figures 2 (a)-(c) shows the number of primary items generated in various databases at different $minSup$ and $minUtil$ values. It can be observed that increase in $minSup$ and/or $minUtil$ results in decrease of primary items, because many items have failed to satisfy the increased $minUtil$ or $minSup$ values.

Figures 3 (a)-(c) shows the number of high utility-frequent itemsets generated by HU-FIMi in various databases at different $minUtil$ and $minSup$ values. It can be observed that increase in $minSup$ and/or $minUtil$ results in decrease of high utility-frequent itemsets. It is because many itemsets have failed to satisfy the increased $minUtil$ or $minSup$. Another important observation in these figures (especially in Yahoo database) is that when $minSup$ is slightly increased from 0 to 0.01(%), there is a significant drop in the number of high utility-frequent itemsets. It is because items with high external utility values were generating too many high utility itemsets by combining with other items when $minSup = 0$. Table 3 presents some of the interesting high utility-frequent itemsets generated in Yahoo database.

Figures 4 (a)-(c) show the runtime requirements of HU-FIMi in various databases at different $minUtil$ and $minSup$ values. It can be observed that increase in $minUtil$ and/or $minSup$ results in decrease of runtime. It is because of the reduction in the number of primary items, which significantly influence the runtime requirements of HU-FIMi. (Memory requirements of HU-FIMi also showed similar affect as the runtime. Due to page limitation, we are unable to present these results.)

7 Conclusions and Future Work

This paper exploited the *utility* order of items and proposed two pruning measures, *cutoff utility* and *suffix utility*, to reduce the computational cost of finding

Table 3. Few interesting itemsets generated in Yahoo database

Itemset	utility (¥)
{face_care:essences, face_care:skin_lotions}	389,476
{ladies:long_sleeve, ladies:knit_sweater:other}	105,994
{ladies:skirt_pants:other, ladies:tops:other}	150,970

the high utility-frequent itemsets. A fast single phase algorithm has also been proposed to find all high utility-frequent itemsets in the data. The experimental results shows that HU-FIMi performs better than EFIM in most cases and we are able to prune infrequent high utility itemsets using *minSup*.

In the literature, HUIM was studied in incremental databases, data streams and uncertain databases. It is interesting to investigate how to extend the proposed pruning measures to discover desired itemsets in all of these databases. The proposed pruning measures consider items' external utility values as positive real numbers. As a part of future, we would like to generalize the proposed pruning measures by taking into account both positive and negative external utility values.

References

1. Fournier-Viger, P., Gomariz, A., Gueniche, T., Soltani, A., Wu, C.W., Tseng, V.S.: Spmf: a java open-source pattern mining library. *The Journal of Machine Learning Research* **15**(1), 3389–3393 (2014)
2. Gan, W., Lin, J.C.W., Fournier-Viger, P., Chao, H.C., Hong, T.P., Fujita, H.: A survey of incremental high-utility itemset mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* **8**(2) (2018)
3. Liu, J., Wang, K., Fung, B.C.: Direct discovery of high utility itemsets without candidate generation. In: *ICDM*. pp. 984–989. IEEE (2012)
4. Liu, Y., Liao, W.k., Choudhary, A.: A two-phase algorithm for fast discovery of high utility itemsets. In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. pp. 689–695 (2005)
5. Md Fazla elahe, K.Z.: Mining high utility itemsets based on multiple minimum support and multiple minimum utility threshold. *International Journal of Database Theory and Application* **10** (2017)
6. Pei, J., Han, J., Wang, W.: Constraint-based sequential pattern mining: the pattern-growth methods. *Journal of Intelligent Information Systems* **28**(2), 133–160 (Apr 2007)
7. Tseng, V.S., Shie, B.E., Wu, C.W., Philip, S.Y.: Efficient algorithms for mining high utility itemsets from transactional databases. *IEEE transactions on knowledge and data engineering* **25**(8), 1772–1786 (2013)
8. Yao, H., Hamilton, H.J., Butz, C.J.: A foundational approach to mining itemset utilities from databases. In: *SIAM*. pp. 482–486 (2004)
9. Zhang, C., Alpanidis, G., Wang, W., Liu, C.: An empirical evaluation of high utility itemset mining algorithms. *Expert Systems with Applications* **101**, 91–115 (2018)
10. Zida, S., Fournier-Viger, P., Lin, J.C.W., Wu, C.W., Tseng, V.S.: Efim: a fast and memory efficient algorithm for high-utility itemset mining. *Knowledge and Information Systems* **51**(2), 595–625 (2017)

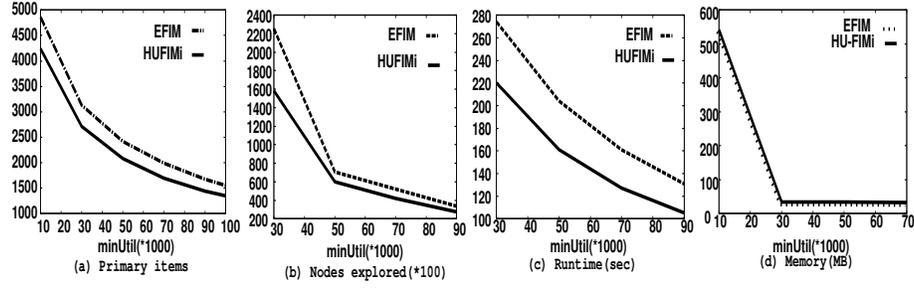


Fig. 1. Performance evaluation of EFIM and HUFIMI algorithms on Yahoo database

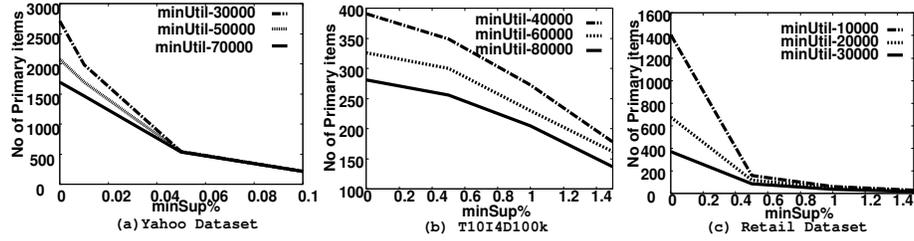


Fig. 2. Primary items generated at different $minSup$ and $minUtil$ values

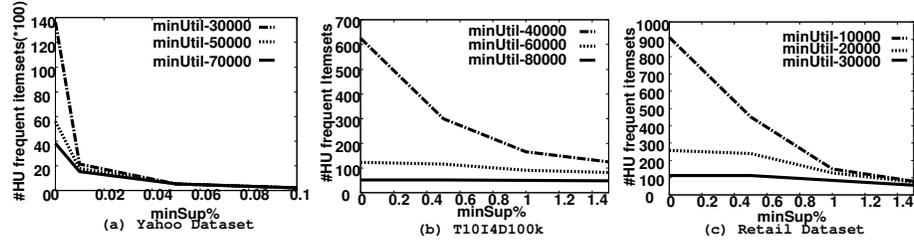


Fig. 3. Itemsets generated at different $minSup$ and $minUtil$ values

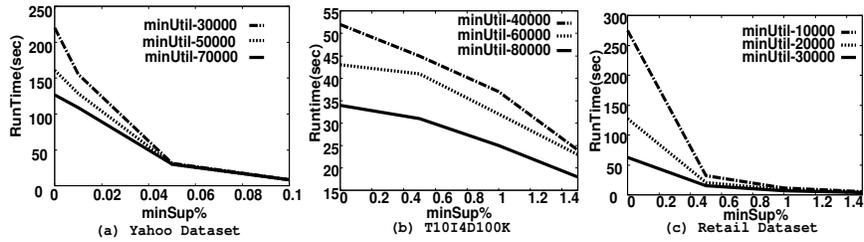


Fig. 4. Runtime of HU-FIMI at different $minSup$ and $minUtil$ values