Efficiently Finding High Utility-Frequent Itemsets using Cutoff and Suffix Utility

R. Uday Kiran[1,2], T. Yashwanth Reddy[3], Philippe Fournier-Viger[4], Masashi Toyoda[2], P. Krishna Reddy[3] and Masaru Kitsuregawa[2,5]

[1]National Institute of Information and Communications Technology, Tokyo, Japan
[2] The University of Tokyo, Tokyo, Japan
[3] International Institute of Information Technology-Hyderabad, Hyderabad, India
[4] Harbin Institute of Technology (Shenzhen), Shenzhen, China
[5]National Institute of Informatics, Tokyo, Japan
{uday_rage,toyoda, kitsure}@tkl.iis.u-tokyo.ac.jp, philfv8@yahoo.com, yashwanth.t@research.iiit.ac.in and pkreddy@iiit.ac.in

Abstract. High utility itemset mining is an important model with many real-world applications. But the popular adoption and successful industrial application of this model has been hindered by the following two limitations: (*i*) computational expensiveness of the model and (*ii*) infrequent itemsets may be output as high utility itemsets. 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 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

High Utility Itemset Model (HUIM) is an important knowledge discovery technique in data mining. It aims to discover all interesting itemsets whose *utility* in a transactional database is no less than a 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 data[1] is:

$$\{Nintendo3Ds\_game,\ Playstation4\_game\}\ \ [utility = 604,231\ ¥].$$

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

The above utility itemset says that the *income* (or *utility*) generated from the simultaneous purchases of 'nintendo3Ds_game' and 'playstation4_game' is 604,231 ¥. This information may be useful to the retailer, because it disproves the general assumption of expecting little revenue from 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 [5]. To reduce the search space in the itemset lattice, most algorithms [2] initially find secondary items by pruning all items that have a *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 construction of projected databases requires a database scan, the computational cost of an high utility itemset mining algorithm depends primarily on the number of primary items. We have observed that finding primary items based on the *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. More importantly, finding high utility itemsets using the *local utility* order makes the model impracticable in many real-world sparse databases.
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 that a significant portion of high utility itemsets generated by HUIM appeared seldomly in the data. It is because the *utility* measure is sensitive to items with high external utility values. Additionally, directly pushing the minimum support constraint into existing HUIM algorithms [2] is not an effective solution to this problem. It is because such algorithms 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 a user-specified *minimum support* ($minSup$) and $minUtil$ values. Using $minSup$ facilitates pruning 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

[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.

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 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), is proposed to find high utility-frequent itemsets efficiently. HU-FIMi is based on EFIM [9]. Experimental results demonstrate that HU-FIMi can discover the desired itemsets efficiently in sparse databases.

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. [7] 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+ [6]) have employed *local utility* to reduce the search space.

Recently, single phase algorithms (e.g. d2HUP [3] and EFIM [9]) 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. [8] 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 efficiently find the desired itemsets.

3 Proposed Model: High Utility-Frequent Itemset

Let $I = \{i_1, i_2, \cdots, i_m\}$, $m \geq 1$, 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,

$UD = \{(i_1, p(i_1)),\ (i_2, p(i_2), \cdots, (i_m, p(i_m))\}$. A **transactional database** is a set of transactions $D = \{T_1, T_2, \cdots, T_n\}$ 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

tid	Items
1	$(a, 2), (b, 3), (f, 2)$
2	$(a, 2), (c, 1), (d, 3), (e, 2)$
3	$(a, 3), (b, 1), (h, 2)$
4	$(c, 2), (d, 3), (e, 1)$
5	$(a, 1), (b, 2), (c, 1)$ $(d, 4), (g, 2)$
6	$(c, 3), (d, 2), (f, 3),$ $(e, 1)$
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, is 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 $u(X, T_c) = \Sigma_{i_j \in X} u(i_j, T_c)$ 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 a database)*** *The utility of an itemset X in the database D, denoted as $u(X) = \Sigma_{T_c \in g(X)} u(X, T_c)$, 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 is $u(ab, T_1) = 1300$ ¥, $u(ab, T_3) = 900$ ¥ and $u(ab, T_5) = 800$ ¥. Therefore, the utility (or *income*) of ab in the database is $u(ab) = 1300 + 900 + 800 = 3000$ ¥.

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 $s(X) \geq minSup$, 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. A high 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

Fournier-Viger et al. [9] 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 a *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* no less than $minUtil$ are considered as primary items. Finally, for each primary item, a projected database consisting of primary and secondary items is constructed and mined recursively 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 $E(\alpha) = \{z,\ z \in I \wedge\ z \succ x,\ \forall\ x \in \alpha\ \}$.*

Example 7. Consider the database from Table 1. Let $\succ$ be the 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 $re(X, T_c) = \Sigma_{i \in T_c \wedge i \succ x \forall x \in X} u(i, T_c)$.*

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

Definition 9. ***(local utility)****. For an itemset α and item $z \in E(\alpha)$, the LocalUtilty of z with respect to α is $lu(\alpha, z) = \Sigma_{T \in g(\alpha, \{z\})}[u(\alpha, T) + re(\alpha, T)]$.*

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

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 is $stu(\alpha, c)$=(((2×200)+(1×200)+(3×400+2×200))+((1×200) + (1×200) + (4×400+2×200)))=4600.

The limitations of EFIM are as follows (i) EFIM finds high utility itemsets without taking into account their *support*. As a result, itemsets with low frequency can be identified as high utility itemsets. (ii) Since creating a projected database for a primary item requires scanning the data, the computational cost of EFIM mostly depends on the number 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 increasing the number of database scans.

5 Proposed Algorithm

HU-FIMi is a single phase algorithm that extends EFIM [9] 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 an monotonic nor anti-monotonic function, 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 consisting of all items whose supersets may be 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,' is $p(g) = 200$. If the user-specified $minSup = 3$, then 'g' should appear at least in three transactions with internal utility of 1. Therefore, the cutoff utility that item '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 [9] can be used to efficiently calculate the TWU, *support* and *utility* of items.
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 the 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 is $u(g) = 400$. 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. All secondary items generated from 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 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 a new pruning measure, called *suffix utility* (see Definition 13), by exploiting items' *utility* descending order. The *suffix utility* facilitates defining 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

Algorithm 2 RecursiveSearch

1: **input** : α: an itemset, $\alpha - D$; the α projected database, $Primary(\alpha)$: the primary items of α, $Secondary(\alpha)$: the secondary items of α, the minutil threshold, the minSup threshold
2: **output**: the set of high utility-frequent itemsets that are extensions of α
3: **for** each *item* $i \in Primary(\alpha)$ **do**
4: $\beta = \alpha \cup \{i\}$;
5: Scan α-D to calculate $u(\beta)$, $s(\beta)$ and create β-D; //uses transaction merging from EFIM
6: **if** $u(\beta) \geq minutil$ && $s(\beta) \geq minSup$ **then**
7: output β
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 β-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(β, β-D, $Primary(\beta)$, $Secondary(\alpha)$, $minutil$, $minSup$);
14: **end if**
15: **end for**

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

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

Property 2. (**Additive property.**) 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 $S = \{i_1, i_2, \cdots, i_k\} \subseteq I$ be an ordered list of secondary items such that $u(i_1) \geq u(i_2) \geq \cdots \geq u(i_k)$. The* suffix utility *of an item $i_j \in S$, denoted as $su(i_j)$, is the 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 is $su(d) = u(b) + u(a) + u(c) + u(e) = 6500$ ¥.

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, $su(i_j) \geq su(i_k)$, where $j < k$ and $i_j, i_k \in S$.

Lemma 1. *Let $S = \{i_1, i_2, \cdots, i_k\} \subseteq I$ be an ordered list of secondary items in utility descending order, i.e., $u(i_1) \geq u(i_2) \geq \cdots \geq u(i_k)$. 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 most cases, the candidate items generated in the previous step form the primary items. However, in a few cases, especially when mining itemsets at low *minUtil* values, suffix utility is inadequate. It is because most secondary items will be generated as 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 computationally expensive step because it needs a 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 employ different ordering of secondary items**.)

Definition 15. *(Primary item). Let C denote the set of candidate items. 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 the 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 the 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. 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 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 the desired itemsets. It can be observed that the proposed algorithm has explored relatively few nodes compared to 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's runtime is shorter than EFIM. The performance improvement of HU-FIMi over EFIM is mainly due to the $suffix\ utility$, which facilitates finding all high utility itemsets with relatively few primary items. Figure 1(d) shows the memory consumed by EFIM and HU-FIMi on the Yahoo database. It is observed that HU-FIMi has consumed slightly more memory than EFIM. It is because HU-FIMi has to additionally store the *support* of each itemset.

Figures 2 (a)-(c) show 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) show 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 an increase in $minSup$ and/or $minUtil$ results in a decrease of high utility-frequent itemsets. It is because many itemsets fail to satisfy the increased $minUtil$ or $minSup$ constraints. Another important observation in these figures (especially in Yahoo database) is that when $minSup$ is slightly increased from 0 to 0.01(%), their 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 when combined with other items for $minSup = 0$. 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 for different $minUtil$ and $minSup$ values. It can be observed that an increase in $minUtil$ and/or $minSup$ results in a decrease in runtime. It is because of the number of primary items is reduced, which significantly influences 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. A few interesting itemsets found in the 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. Experimental results shows that HU-FIMi outperforms EFIM in most cases and is 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 the desired itemsets in such databases. The proposed pruning measures consider items' external utility values as positive real numbers. In future work, we would like to generalize the proposed pruning measures by taking into account both positive and negative external utility values.

8 Acknowledgements

We would like to thank Yahoo Japan Corporation for providing the retail transaction data.

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. 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)
6. Tseng, V.S., Shie, B.E., Wu, C.W., Yu, P.S.: Efficient algorithms for mining high utility itemsets from transactional databases. IEEE transactions on knowledge and data engineering **25**(8), 1772–1786 (2013)
7. Yao, H., Hamilton, H.J., Butz, C.J.: A foundational approach to mining itemset utilities from databases. In: SIAM. pp. 482–486 (2004)
8. Zhang, C., Almpanidis, G., Wang, W., Liu, C.: An empirical evaluation of high utility itemset mining algorithms. Expert Syst. with Appl. **101**, 91–115 (2018)
9. 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)

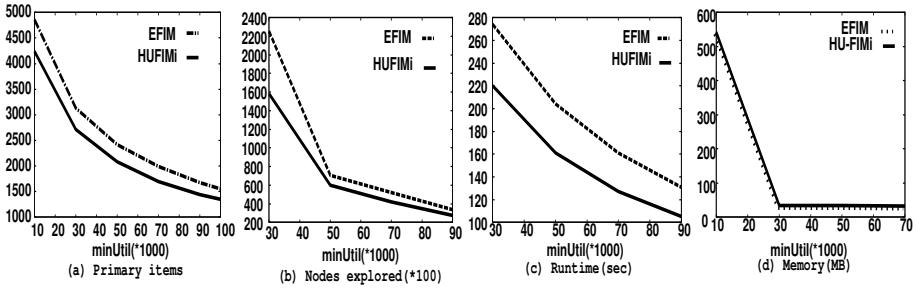

Fig. 1. Performance evaluation of EFIM and HUFIM algorithms on Yahoo database

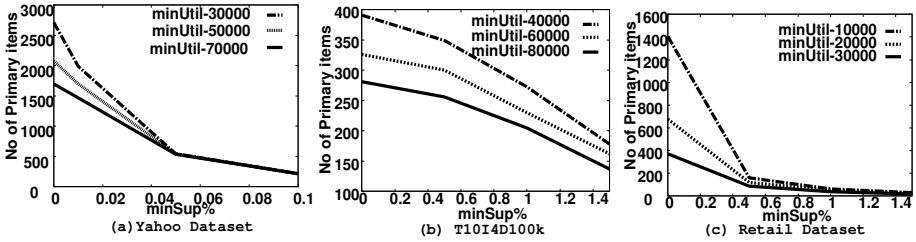

Fig. 2. Primary items generated at different *minSup* and *minUtil* values

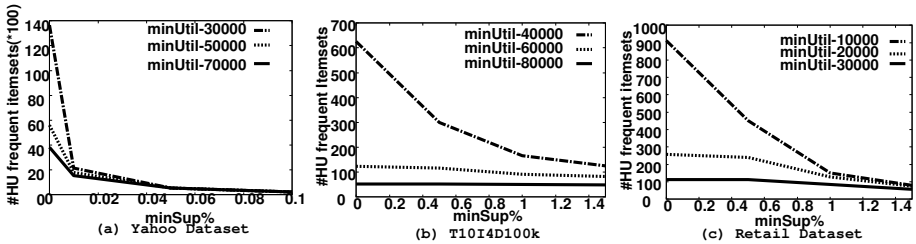

Fig. 3. Itemsets generated at different *minSup* and *minUtil* values

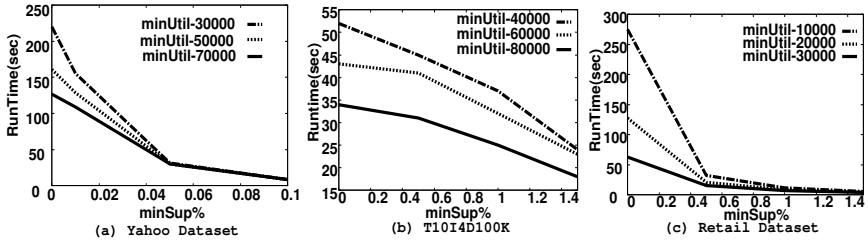

Fig. 4. Runtime of HU-FIMi at different *minSup* and *minUtil* values