An Efficient Algorithm for Mining High-Utility Itemsets with Discount Notion

Ruchita Bansal, Siddharth Dawar(✉), and Vikram Goyal

Indraprastha Institute of Information Technology-Delhi (IIIT-D), New Delhi, India
{ruchita13104,siddharthd,vikram}@iiitd.ac.in

Abstract. High-utility itemset mining has attracted significant attention from the research community. Identifying high-utility itemsets from a transaction database can help business owners to earn a better profit by promoting the sales of high-utility itemsets. The technique also finds applications in web-click stream analysis, biomedical data analysis, mobile E-commerce etc. Several algorithms have been proposed to mine high-utility itemsets from a transaction database. However, these algorithms assume that items have a constant profit associated with them and don't embed the notion of discount into the utility-mining framework. In this paper, we integrate the notion of discount in state-of-the-art utility-mining algorithms and propose an algorithm for efficiently mining high-utility itemsets. We conduct extensive experiments on real and synthetic datasets and our results show that our proposed algorithm outperforms the state-of-the-art algorithms in terms of total execution time and number of itemsets that need to be explored.

1 Introduction

High-utility itemset mining finds patterns from a database which have their utility value no less than a given minimum utility-threshold. In utility-itemset mining, a utility function is generally defined to measure the importance of an itemset and varies according to an application domain. For example, in a retail store, a utility function can measure the profit made by the store by selling the items in the itemset together over a period of time. Recently, the high-utility itemset mining research has up-surged due to its wide applicability to different applications such as identifying fraudulent credit card transactions, network intrusions, medicine [1], molecular biology etc. Utility-mining has also been applied with other mining techniques like sequential-pattern mining [2] and episode-pattern mining [3]. One of the main reason of interest in the topic is due to its model expressiveness in terms of capturing the relevance and multiplicity of individual items within a transaction, which were the shortcomings in the case of frequent itemset mining (FIM). The FIM considers the frequency of itemsets to define the importance of an itemset and works on the model of presence and absence of an item only.

The majority of the utility mining algorithms consider that the items have constant profits associated them and do not take into account various discount

N. Kumar and V. Bhatnagar (Eds.): BDA 2015, LNCS 9498, pp. 84–98, 2015.
DOI: 10.1007/978-3-319-27057-9_6

schemes prevalent in the retail-market domain. However, in real life scenarios, shopkeepers often give discounts on the bulk purchase of items to improve their sales and earn a better profit. Discounts are also given on items which are close to being expired. Online shopping sites like Flipkart and Snapdeal provide discounts on the bulk purchase of items which varies for different items. Discounts are also given on the supply side of the retail chain. For example, Wal-Mart is able to offer low prices in part because it buys large volumes of goods from suppliers at cheaper rates. Suppliers often bid low prices in order to get a large amount of business which Wal-Mart promises. Mining of high-utility itemsets may result in the generation of false positives if the discount notion is not included. We propose to integrate the discount notion in the process of high-utility mining so as to get the correct high-utility itemsets as the output. Specifically, algorithms proposed by Dawar et al. [4] and Viger et al. [5] are two state-of-the-art algorithms for mining high-utility itemsets. We show how to embed the notion of discount in these algorithms and avoid false positives in the result set.

The algorithms on high-utility itemset mining can be classified into two different paradigms, tree-based algorithms [4,6] and vertical mining algorithms [7]. The tree-based algorithms mainly work in two phases. In the first phase, they find candidate high-utility itemsets, which are then verified in the second phase. The advantage of tree-based approaches is that the tree data structure is a compact representation of the complete transaction database and allows mining of candidate high-utility itemsets quickly. However, the verification time taken by these algorithms increases with the number of candidates generated. The vertical mining algorithms use an Inverted-list like data structure for its working. The algorithms in this category basically work similar to Apriori [8] and generate k-length high-utility itemsets by intersecting (joining) the $k-1$ itemsets. Basically, these algorithms first generate all the singleton high-utility itemsets and then proceed to the generation of pairs, triplets and so on. However, solutions based on vertical mining approach are simple and have shown to perform better as compared to tree-based approaches. But, the join operation cost is generally higher for small-size itemsets as compared to join operation cost for large-size itemsets. It is due to the size of lists associated with small itemsets being more than size of lists associated with large itemsets. On the other hand, it is efficient to perform join for itemsets with small size Inverted-list.

In order to avoid the costly join operation for short itemsets in the case of Inverted-list based approaches and avoid the costly operation of verifying the candidates in the case of tree-based approaches, we propose an algorithm which generates high-utility patterns without generating any candidate. Basically, our proposed approach combines the techniques used in tree-based and vertical mining based algorithm. The idea is to start with a tree-based recursive algorithm and traverse the tree until there is a possibility of high-utility itemset being generated. The algorithm then switches to vertical mining algorithm. We, in this case, study the performance gained by combining UP-Hist tree and FHM approaches.

Our research contributions can be summarized as follows:

- We show how the notion of discount can be embedded in the state-of-the-art high-utility itemsets mining algorithms to efficiently retrieve the correct set of patterns.
- We propose an algorithm, namely UP-Hist FHM discount, which combines two state-of-the-art algorithms, UP-Hist Growth and FHM.
- We conduct extensive experiments on real as well as synthetic datasets and the results demonstrate that our proposed algorithm outperforms the state-of-the-art algorithms individually in terms of total execution time and the number of itemsets explored.

2 Related Work

Frequent-itemset mining [8,9] has been studied extensively in the literature. Agrawal et al. [8] proposed an algorithm named Apriori, for mining association rules from market-basket data. Their algorithm was based on the downward closure property [8]. The downward closure property states that every subset of a frequent itemset is also frequent. Park et al. [10] proposed a hash-based algorithm for mining association rules which generates less number of candidates compared to Apriori algorithm. Zaki et al. [11] proposed an algorithm, namely ECLAT, for mining association rules which used itemset clustering to find the set of potentially maximal frequent itemsets. Han et al. [9] proposed a pattern-growth algorithm to find frequent itemsets by using FP-tree data structure. Vu et al. [12] proposed an algorithm namely FEM, which combined the FP-Growth and ECLAT algorithm for mining frequent patterns. However, frequent-itemset mining algorithms can't be used to find high-utility itemsets as it is not necessarily true that a frequent itemset is also a high-utility itemset in the database. On the other hand, mining high-utility patterns is challenging compared to the frequent-itemset mining, as there is no downward closure property [8] like we have in frequent-itemset mining scenario.

Several algorithms have also been proposed to find high-utility itemsets. Liu et al. [13] proposed a two-phase algorithm which generates candidate high-utility itemsets in the first phase and verification is done in the second phase. Ahmed et al. [14] proposed another two-phase algorithm, which uses a data structure named IHUP-Tree, to mine high-utility patterns incrementally from dynamic databases. The problem with the above-mentioned algorithms is the generation of a huge amount of candidates in the first phase which leads to longer execution times. In order to reduce the number of candidates, Tseng et al. [6] proposed a new data structure called UP-Tree and algorithms, namely UP-Growth [6] and UP-Growth+ [15]. The authors proposed effective strategies like DGU, DGN, DLU and DLN to compute better utility estimates. In order to reduce the number of candidates generated in the first phase by UP-tree based algorithms, Dawar et al. [4] proposed another data structure UP-Hist tree and better utility estimates. Liu et al. [7] proposed a new data structure named utility-lists and an algorithm *HUI-Miner* for mining high-utility itemsets. The algorithm avoids the

costly generation and verification of candidates. However, the joining of utility-lists of an itemset to produce a new itemset is a costly operation. In order to reduce the number of join operations, Viger et al. [5] proposed a novel strategy EUCP(Estimated Utility Co-occurrence Pruning) to prune itemsets without performing the join operation. Recently, Li et al. [16] proposed several strategies to embed discount notion in the utility mining framework. Discount strategies like "buy 2 get 1 free", "buy 1 get the second at 70 % discount" were considered. They proposed a three-phase level-wise algorithm to mine high-utility itemsets. In this paper, we propose an algorithm, which is efficient compared to the state-of-the-art algorithm for mining high-utility itemsets with discount strategies.

3 Background

In this section, we present some definitions given in the earlier works and describe the problem statement formally. We also discuss the data structures and state-of-the-art algorithms for mining high-utility itemsets briefly.

3.1 Preliminary

We have a set of m distinct items $I = \{i_1, i_2, ..., i_m\}$, where each item has a profit $pr(i_p)$ $(external\ utility)$ with respect to number of quantities. An itemset X of length k is a set of k items $X = \{i_1, i_2, ..., i_k\}$, where for $j \in 1.....k$, $i_j \in I$. A transaction database $D = \{T_1, T_2, ..., T_n\}$ consists of a set of n transactions, where every transaction has a subset of items belonging to I. Every item i_p in a transaction T_d has a quantity $q(i_p, T_d)$ associated with it. Below we define how utility of an item, an itemset can be computed in the context of a transaction.

Definition 1 ***(Utility of an item in a transaction).*** *The utility of an item i_p in a transaction T_d is denoted as $u(i_p, T_d)$ and defined as the product of the profit of the item and its quantity in the transaction i.e. $u(i_p, T_d) = q(i_p, T_d) * pr(i_p)$.*

Definition 2 ***(Utility of an itemset in a transaction).*** *The utility of an itemset X in a transaction T_d is denoted as $u(X, T_d)$ and defined as $\sum_{X \subseteq T_d \wedge i_p \in X} u(i_p, T_d)$.*

We also define a utility of a transaction as similar to an itemset over a transaction as given below.

Definition 3 ***(Utility of transaction).*** *The utility of a transaction T_d is denoted as $TU(T_d)$ and defined as $\sum_{i_p \in T_d} u(i_p, T_d)$.*

Let us consider the example database shown in Table 1 and the profit associated with each item in Table 2. The utility of item $\{A\}$ in $T_3 = 5$ and the utility of itemset $\{A, B\}$ in T_3 denoted by $u(\{A, B\}, T_3) = u(A, T_3) + u(B, T_3) = 5 + 6 = 11$. The transaction utility of every transaction is shown in Table 1.

Table 1. Example Database

TID	Transaction	TU
T_1	$(A:1)\,(C:1)\,(D:1)$	9
T_2	$(A:2)\,(C:6)\,(E:2)\,(G:5)$	67
T_3	$(A:1)\,(B:2)\,(C:1)\,(D:6)\,(E:1)\,(F:5)$	40
T_4	$(B:4)\,(C:3)\,(D:3)\,(E:1)$	29
T_5	$(B:2)\,(C:2)\,(E:1)\,(G:2)$	29

Table 2. Profit Table

Item	Profit
A	5
B	3
C	2
D	2
E	5
F	2
G	7

Definition 4 ***(Utility of an itemset in Database).*** *The utility of an itemset X in database D is denoted as $u(X)$ and defined as $\sum_{X \subseteq T_d \wedge T_d \in D} u(X, T_d)$.*

For example, $u(B,C) = u(\{B,C\},T_3) + u(\{B,C\},T_4) + u(\{B,C\},T_5) = 8 + 18 + 10 = 36$.

Definition 5 ***(High-utility itemset).*** *An itemset is called a high-utility itemset if its utility is no less than a user-specified minimum threshold denoted by min_util.*

For example, $u(C,E) = u(\{C,E\},T_2) + u(\{C,E\},T_3) + u(\{C,E\},T_4) + u(\{C,E\},T_5) = 22 + 7 + 11 + 9 = 49$. If $min\_util = 40$, then $\{C,E\}$ is a high-utility itemset. However, if $min\_util = 50$, then $\{C,E\}$ is a low-utility itemset.

Definition 6 ***(Problem Statement).*** *Given a transaction database D and a minimum utility threshold min_util, the aim is to find all the itemsets which have high-utility.*

The high-utility itemsets at minimum utility threshold 50 are $\{CG\}$:65, $\{EG\}$:64, $\{ACG\}$:57, $\{AEG\}$:55, $\{BCE\}$:51, $\{CEG\}$:80, $\{ACEG\}$:67 and $\{BCDE\}$:54. We now describe the concept of transaction utility and transaction weighted downward closure (TWDC) [17].

Definition 7 ***(TWU of an itemset).*** *Transaction-weighted utility of an itemset X is the sum of the transaction utilities of all the transactions containing X, which is denoted as $TWU(X)$ and defined as $\sum_{X \subseteq T_d \wedge T_d \in D} TU(T_d)$.*

Definition 8 ***(High TWU itemset).*** *An itemset X is called a high-transaction-weighted utility itemset (HTWUI), if $TWU(X)$ is no less than min_util.*

Property 1 (Transaction-weighted downward closure). For any itemset X, if X is not a (HTWUI), any superset of X is not a HTWUI.

For example, $TU(T_1) = u(\{ACD\},T_1) = 9; TWU(\{A\}) = TU(T_1) + TU(T_2) + TU(T_3) = 116$. If $min\_util = 110$, $\{A\}$ is a HTWUI. However, if $min\_util = 120$, $\{A\}$ and any of its supersets are not HTWUIs.

3.2 UP-Hist Tree and UP-Hist Algorithm

UP-Hist Growth is a tree-based algorithm that uses the UP-Hist tree for mining high-utility itemsets. The UP-Hist tree is created in two database scans. In the first scan of the database, *TWU* of individual items is computed. Items which have their *TWU* values less than the minimum threshold are identified and removed. The remaining items in the transactions are sorted in descending order of their *TWU* value and inserted to construct a global UP-Hist tree in the second scan. The root of the tree is a special empty node which points to its child nodes. Every other node N in UP-Hist tree [4] consists of a name $N.item$, overestimated utility $N.nu$, support count $N.count$, a histogram of item quantities (explained later), a pointer to the parent node $N.parent$ and a pointer $N.hlink$ to the node which has the same name as $N.name$. The support count of a node N along a path is the number of transactions in the database that contain the itemset consisting of items on the path from the root to that node. In order to facilitate efficient traversal of the tree, a header table is also maintained. The header table has three columns, *Item*, *TWU* and *Link*. The nodes in a UP-Hist tree along a path are maintained in descending order of their *TWU* values in the header table. All nodes with the same label are stored in a linked list and the link pointer in the header table points to the head of the list. The histogram associated with every node of the UP-Hist tree is defined below.

Definition 9 ***(Histogram).*** *A histogram h is a set of pairs $\langle q_i, num_i \rangle$, where q_i is an item quantity and num_i is the number of transactions that contain q_i copies of an item.*

The histogram associated with each node helps in computing the minimum and maximum quantity estimates of the node as defined below.

Definition 10 ***(minC).*** *Let h be a histogram, associated with an item-node N_i, consisting of n, $(1 \leq i \leq n)$ pairs $< q_i, num_i >$, sorted in* ***ascending order*** *of q_i. $minC(N_i, s)$ returns the sum of item-copies of k entries of h, i.e., $minC(N_i, s) = \sum_1^k q_i$, such that k is the minimal number fulfilling $k \leq \sum_1^k num_i$.*

Definition 11 ***(maxC).*** *Let h be a histogram, associated with an item-node N_i, consisting of n, $(1 \leq i \leq n)$ pairs $< q_i, num_i >$, sorted in* ***descending order*** *of q_i. $maxC(N_i, s)$ returns the sum of item-copies of k entries of h, i.e., $maxC(N_i, s) = \sum_1^k q_i$, such that k is the minimal number fulfilling $k \leq \sum_1^k num_i$.*

For example, the histogram of item C is $h = \{< 1, 2 >, < 2, 1 >, < 3, 1 >, < 6, 1 >\}$ and let support be 3. $minC(C, 3)$ and $maxC(C, 3)$ is 3 and 11 respectively. These quantity estimates are used by the algorithm to finally compute better estimates for the lower-bound and upper-bound value of any itemset. The readers can refer to [4] for more details of the algorithm.

3.3 Utility-List Data Structure and FHM Algorithm

FHM [5] is a vertical data mining algorithm that uses a utility-list data structure for mining high-utility itemsets. A utility-list associated with an itemset I is a list of triples storing three columns of information: TID, Iutils and Rutils. TID is a transaction identifier. Iutils(I, T_i) is the exact utility of itemset I in the transaction T_i. Rutils(I, T_i) is the aggregate utility value of items which occur after itemset I in transaction T_i. FHM algorithm assumes that items in a transaction are sorted in ascending order of their TWU values. For example, the utility-list of items $\{A\}$ and $\{B\}$ for our example database is shown in Fig. 1. FHM works similar to HUI-Miner algorithm in a level-wise manner. The algorithm joins two $\{k-1\}$-length itemsets to get a $\{k\}$-length itemset. For example, the utility-list of itemset $\{AB\}$ constructed from the intersection of utility-list of item $\{A\}$ and $\{B\}$ consists of single tuple $< 3, 11, 30 >$ only.

TID	Iutils	Rutils
1	5	2
2	10	22
3	5	7

TID	Iutils	Rutils
3	6	12
4	12	11
5	6	23

Fig. 1. Utility-list of item $\{A\}$ and $\{B\}$

3.4 Three-Phase Algorithm for Mining High-Utility Itemsets with Discount Strategies

Recently, Li et al. [16] proposed a three-phase algorithm for mining high-utility itemsets from a transaction database by applying several discount strategies. They propose to use rules to specify discount strategies and discussed four rules; "buy 1 with 50 % discount", "buy 2 get 1 free", "buy 1 get the second at 70 % discount" and "zero discount or no discount".

To mine correct set of high-utility itemsets with a discount notion, they incorporate all applicable discounts while computing of TWU of singleton itemsets. The $\{1\}$-itemset with their TWU less than the threshold are pruned immediately. In the second phase, a level-wise search is performed to find all the candidate high-utility itemsets. However, they do not consider any discount related information afterwards and their utility estimates are very loose. Due to which, their approach ends up in the generation of many candidate itemsets. In the third phase, the exact utility of candidate-itemsets is computed to find the actual high-utility itemsets.

4 Integration of Discount Notion in State-of-the-art Algorithms

In this section, we will discuss how trivial it is to introduce the discount notion in state-of-the-art algorithms, UP-Hist Growth and FHM.

4.1 UP-Hist Discount Algorithm

The utility value estimates are the places where if the discount notion is incorporated then it may help to compute better utility estimates. The tight estimates will result into improved pruning of useless search space. The UP-Hist algorithm computes TWU, Lower-bound and Upper-bound utility values of itemsets while exploring the search space for finding high-utility itemsets. The incorporation of various discounts schemes on items while computing the TWU value of singleton itemsets is straight forward, i.e., apply discount rules while computing the TU value of a transaction. We embed the notion of discount by assuming that different profits are associated with different quantities of an item x in the database. The discount strategies specified by Li et al. [16] can also be represented by a quantity-profit table as shown in Table 3.

Table 3. Quantity Profit Table

Item	1	2	3	4	5	6
A	5	7	11	13	13	13
B	3	4	5	6	10	10
C	2	3	4	6	8	10
D	2	3	4	6	8	10
E	5	8	13	17	21	21
F	2	3	4	7	9	9
G	7	8	18	25	30	30

Definition 12 ***(Maximum and Minimum profit per unit item of an item x).*** *Let x be an item in database D. Let $minq(x)$ and $maxq(x)$ be the minimum and maximum quantity associated with item x in D. The minimum quantity of item x in the database. Maximum profit per unit item of x denoted by $max\_pr(x)$ is defined as*

$$max\_pr(x) = pr_x(minq(x))/minq(x)$$

$pr_x(minq(x))$ is the profit of item x at $minq(x)$ quantity from the quantity profit table. Similarly, $min\_pr(x)$ is defined as

$$min\_pr(x) = pr_x(maxq(x))/maxq(x)$$

We now show how discounts rules can be considered to compute lower-bound and upper-bound estimates.

Definition 13 ***(Maximum utility of an item in a set of transactions with support s).*** *Let x be an item with support $total\_support(x)$, utility $u(x)$ in database D. $minq(x)$ is the minimum quantity of item x in the database. The maximum utility of x for s transactions is denoted by $MaxU(x, s)$ is defined as*

$$MaxU(x, s) = u(x) - (minq(x) * min\_pr(x)).$$

Definition 14 ***(Minimum utility of an item in a set of transactions with support s).*** *Let x be an item with support $total\_support(x)$, utility $u(x)$ in database D. $maxq(x)$ is the maximum quantity of item x in the database. The minimum utility of x for s transactions is denoted by $MinU(x, s)$ is defined as*

$$MinU(x, s) = u(x) - (maxq(x) * max\_pr(x))$$

Using the upper-bound and lower-bound utility values of individual items, we can compute the lower-bound and upper-bound utility value of an itemset as given below.

Definition 15 ***(Upper-bound utility).*** *Given an itemset $I =< a_1, a_2, ..., a_k >$ corresponding to a path in UP-Hist tree , with support count s, the upper-bound utility of itemset I denoted by $ub(I)$ is defined as*

$$ub(I) = \sum_{i=1}^{k} min(maxC(a_i, s) * max\_lpr(a_i), MaxU(a_i, s))$$

$max\_lpr(a_i)$ is the maximum profit per unit item of a_i.

Definition 16 ***(Lower-bound utility).*** *Given an itemset $I =< a_1, a_2, ..., a_k >$ corresponding to a path in UP-Hist tree , with support count s, the lower bound utility value of itemset I denoted by $lb(I)$ is defined as*

$$lb(I) = \sum_{i=1}^{k} max(minC(a_i, s) * min\_lpr(a_i), MinU(a_i, s))$$

$min\_lpr(a_i)$ is the minimum profit per unit item of a_i.

Claim 1. *The utility values of an itemset I are correct lower bound and upper bound estimates of the exact utility of I.*

4.2 FHM Discount Algorithm

The FHM algorithm constructs the utility-list of singleton items and explores the search-space in a level-wise manner. The utility-list data structure keeps the information of utility of an itemset transaction-wise. Therefore, once discount rules are applied for individual items that information remains in the utility list for each transaction. During join of two $\{k-1\}$ length itemsets, computation of exact-utility and remaining-utility uses utility value of each node (transaction) in the intersection list. Therefore, discount rules once applied over each transaction are carried forward in the case of a vertical mining-based approach like FHM.

5 Mining High-Utility Itemsets

In this section, we present our algorithm, UP-Hist FHM discount, that mines high-utility itemsets from a transaction database and incorporates discount notion. We will also illustrate the working of our algorithm with an example.

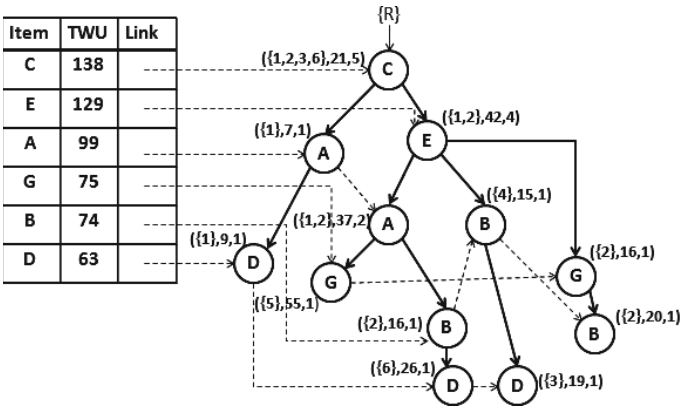

Fig. 2. Global UP-Hist Tree

Our algorithm is recursive and requires two scans of the database. In the first scan, transactions are scanned to compute the TWU values of each item. Items which have their TWU less than the utility threshold parameter are called as unpromising items and removed from further consideration. In the next scan of the database, items in every transaction are sorted in decreasing order of their TWU values. The transactions are then inserted one-by-one to construct the global UP-Hist tree. The utility-list of singleton items is also created in this step.

Each item i is picked from the header table in a bottom-up manner. The sum of node utility is computed by traversing the linked list associated with item i and if the sum is no less than the utility threshold, the upper-bound utility of the itemset including item i is computed. However, if the sum value for the node is less, the current itemset including item i as well as any superset itemset can not be of high-utility. Therefore, no processing is done further for the itemset. In the other case, the upper-bound utility is checked with respect to the threshold parameter. The algorithm switches to FHM strategy if the estimate of the itemset is no less than the threshold, i.e., the utility-list of the itemset is created and FHM algorithm is invoked. Else, the local tree is created and the UP-Hist algorithm is called recursively. We now present an example to illustrate the working of our proposed algorithm. We consider the example database shown in Table 1 with a quantity profit table as shown in Table 3. Let the utility threshold be 36. The global UP-Hist tree is shown in Fig. 2. Since, the header table is processed in a bottom-up manner, item $\{D\}$ is processed first.

The linked list associated with item $\{D\}$ is traversed and the sum of node utilities will be computed. The sum of node utilities is greater than the threshold. Therefore, item D is processed further. The upper-bound utility of item D is 16. Since, the upper bound utility value is less than threshold, a local tree for prefix D is created using paths $< CA >$: 9, $< CEAB >$: 26 and $< CEB >$: 19. The set of these paths is called Conditional Pattern Base (CPB). The TWU of each item

Algorithm 1. UP-Hist FHM discount(T_x, H_x, X)

Input: A UP-Hist tree T_x, a header table H_x, an itemset X, a minimum utility threshold min_util.
Output: All candidate high-utility itemsets in T_x.

1: **for** each entry $\{i\}$ in H_x **do**
2: Compute $node.nu\_sum$ by following the links from the header table for each item $\{i\}$. Also, compute the upper bound utility of item $\{i\}$ denoted by $UB(\{i\})$
3: **if** (**then** $node.nu\_sum(i) \geq min\_util$)
4: **if** (**then** $UB_{sum}(a\_i) \geq min\_util$)
5: Construct $I = prefix \cup i$ and its utility-list. Construct the utility-list of I-1 extensions (ULs) and call FHM(Y,item,ULs,threshold)
6: **else**
7: Construct the CPB of $I = X \cup i$.
8: **end if**
9: Put local promising items in $\{I\} - CPB$ into H_I and apply DLU, DLN. Insert every reorganized path into T_I.
10: **if** $T\_I \neq null$ **then**
11: Call UP-Hist FHM discount(T_I,H_I,I)
12: **end if**
13: **end if**
14: **end for**

in the CPB is computed similar to original database and unpromising items are removed. For our example, Item A is an unpromising item and hence removed. The transactions are reorganized and inserted to form the local tree of item $\{D\}$. In the next recursive invocation, item $\{B\}$ from the local header of $\{D\}$ is processed. Like the previous step, the sum of node-utility for item $\{B\}$ is greater than the threshold and the upper bound utility of itemset $\{BD\}$ is 25, which is less than the minimum threshold. The local tree of item $\{BD\}$ prefix is created and the algorithm is called recursively. Next item in the header table is $\{E\}$. The upper-bound utility value of itemset $\{EBD\}$ is 40, which is greater than the threshold. Therefore, the algorithm now constructs a utility-list for itemset $\{BED\}$ and switches to FHM strategy. The FHM algorithm computes the exact utility of itemset $\{BED\}$, which is 34. The algorithm explores the supersets of $\{EBD\}$ to find high-utility itemsets. After FHM completes its execution, the execution proceeds with the UP-Hist Growth algorithm. The complete set of high-utility itemsets returned by the algorithm is: $\{BCDE\}$:40, $\{BCE\}$:38, $\{G\}$:38, $\{AG\}$:37, $\{AEG\}$:45, $\{ACEG\}$:55, $\{ACG\}$:47, $\{EG\}$:51, $\{CEG\}$:64, $\{CG\}$:51, $\{ACE\}$:37 and $\{CE\}$:42.

6 Experiments and Results

In this section, we compare the performance of our proposed algorithm against the state-of-the-art algorithms UP-Hist Growth [4] and FHM [5]. We integrated our model in FHM and UP-Hist Growth to make them comparable with our

Table 4. Characteristics of Real Datasets

Dataset	#T_x	Avg. length	#Items	Type
Kosarak	9,90,002	8.1	41270	Sparse
Retail	88,162	10.3	16470	Sparse
Accidents	3,40,183	33.8	468	Dense
Connect	67,557	43	129	Dense

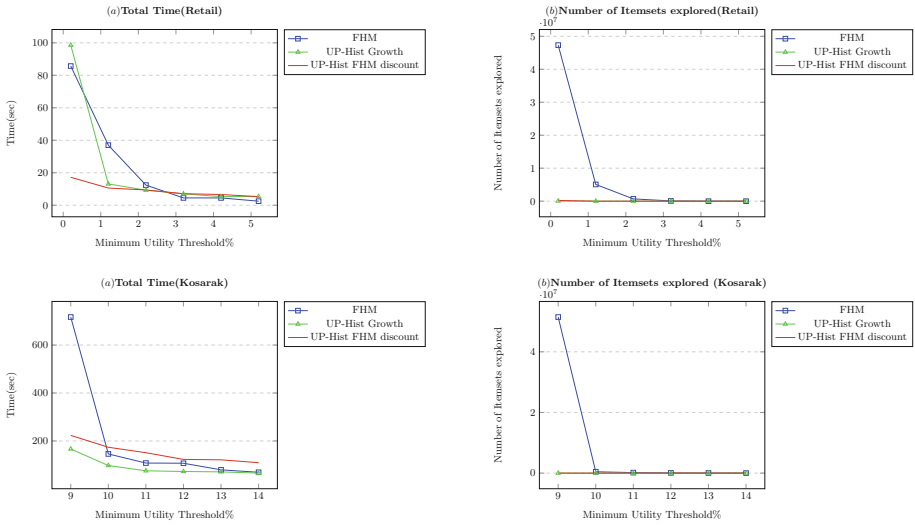

Fig. 3. Performance Evaluation on Sparse Datasets

proposed algorithm. We also implemented the three-phase algorithm integrating discount strategies [16]. However, we don't report the results of the three-phase algorithm as the execution didn't stop for three days on dense datasets. The algorithm gave out of memory error when executed on sparse datasets. We conduct experiments on various real and synthetic datasets. The description of the real datasets is shown in Table 4. We implemented all the algorithms in Java with JDK 1.7 on a Windows 8 platform. The experiments were performed on an Intel Xeon(R) CPU=26500@2.00 GHz with 64 GB RAM. All real datasets were obtained from FIMI Repository [18]. The quantity information for items was chosen randomly from 1 to 5. The external utility values were generated between 1 to 1000 using log-normal distribution. We compared the performance of the algorithms on the basis of total execution time as well as the number of itemsets explored. In our experiments, the utility values are expressed in terms of percentage. For each dataset, we find the utility threshold above which there are no high-utility itemsets and use it as a reference to express other threshold values in percentage. The results on sparse datasets are shown in Fig. 3.

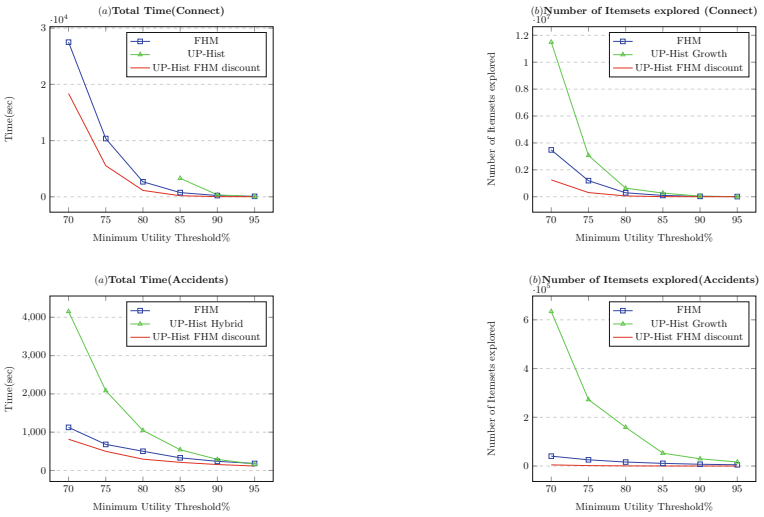

Fig. 4. Performance Evaluation on Dense Datasets

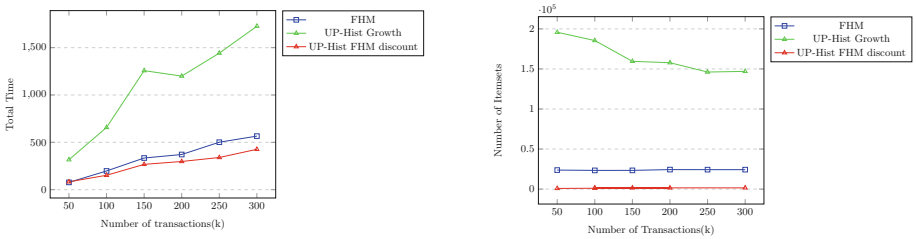

Fig. 5. Scalability on Accidents Dataset

The results show that our proposed algorithm beats FHM in terms of total execution time and number of itemsets explored. We observe that the difference between the runtime of the algorithms becomes marginal at high threshold value.

The results on dense datasets are shown in Fig. 4. The results show the better performance of our algorithm at lower threshold values, especially on Connect and Accidents dataset. UP-Hist performs worst on dense datasets as it generates a lot of candidate itemsets which need to be verified later. We are unable to report the execution time on Connect dataset at lower threshold values as UP-Hist algorithm didn't stop execution for more than 10 h.

We also conduct experiments to evaluate the scalability of our algorithm on Accidents dataset by varying the number of transactions in the dataset and the result is shown in Fig. 5. The result shows that performance of our algorithm improves with an increase in the number of transactions.

7 Conclusion and Future Work

In this paper, we integrate the notion of discount in the state-of-the-art algorithms and propose an algorithm to mine high-utility itemsets. We conduct extensive experiments on various real and synthetic datasets and the results confirm the superior performance of our algorithm compared to the state-of-the-art algorithms in terms of total execution time and the number of itemsets explored.

Acknowledgement. Third author will like to acknowledge the support provided by ITRA project, funded by DeITy, Government of India, under grant with Ref. No. ITRA/15(57)/Mobile/HumanSense/01.

References

1. Medici, F., Hawa, M.I., Giorgini, A., Panelo, A., Solfelix, C.M., Leslie, R.D., Pozzilli, P.: Antibodies to gad65 and a tyrosine phosphatase-like molecule ia-2ic in filipino type 1 diabetic patients. Diabetes Care **22**(9), 1458–1461 (1999)
2. Yin, J., Zheng, Z., Cao, L.: Uspan: an efficient algorithm for mining high utility sequential patterns. In: Proceedings of the 18th ACM SIGKDD, pp. 660–668 (2012)
3. Wu, C.-W., Lin, Y.-F., Yu, P.S., Tseng, V.S.: Mining high utility episodes in complex event sequences. In: Proceedings of the 19th ACM SIGKDD, pp. 536–544 (2013)
4. Dawar, S., Goyal, V.: Up-hist tree: An efficient data structure for high utility pattern mining from transaction databases. In: International Database Engineering and Applications Symposium (2015)
5. Fournier-Viger, P., Wu, C.-W., Zida, S., Tseng, V.S.: FHM: faster high-utility itemset mining using estimated utility co-occurrence pruning. In: Andreasen, T., Christiansen, H., Cubero, J.-C., Raś, Z.W. (eds.) ISMIS 2014. LNCS, vol. 8502, pp. 83–92. Springer, Heidelberg (2014)
6. Tseng, V.S., Wu, C.-W., Shie, B.-E., Yu, P.S.: Up-growth: an efficient algorithm for high utility itemset mining. In: ACM SIGKDD, pp. 253–262 (2010)
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, pp. 55–64 (2012)
8. Agrawal, R., Srikant, R., et al.: Fast algorithms for mining association rules. In: Proc. 20th VLDB, vol. 1215, pp. 487–499 (1994)
9. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. ACM SIGMOD **29**, 1–12 (2000)
10. Park, J.S., Chen, M.-S., Yu, P.S.: An effective hash-based algorithm for mining association rules, vol. 24 (1995)
11. Zaki, M.J., Parthasarathy, S., Ogihara, M., Li, W., et al.: New algorithms for fast discovery of association rules. In: KDD, vol. 97, pp. 283–286 (1997)
12. Vu, L., Alaghband, G.: A fast algorithm combining fp-tree and tid-list for frequent pattern mining. In: Proceedings of Information and Knowledge Engineering, pp. 472–477 (2011)
13. Liu, Y., Liao, W., Choudhary, A.K.: A two-phase algorithm for fast discovery of high utility itemsets. In: Ho, T.-B., Cheung, D., Liu, H. (eds.) PAKDD 2005. LNCS (LNAI), vol. 3518, pp. 689–695. Springer, Heidelberg (2005)

14. Ahmed, C.F., Tanbeer, S.K., Jeong, B.-S., Lee, Y.-K.: Efficient tree structures for high utility pattern mining in incremental databases. IEEE TKDE **21**(12), 1708–1721 (2009)
15. Tseng, V.S., Shie, B.-N., Wu, C.-W., Yu, P.S.: Efficient algorithms for mining high utility itemsets from transactional databases. IEEE TKDE **25**(8), 1772–1786 (2013)
16. Li, Y., Zhang, Z.H., Chen, W.B., Min, F.: Mining high utility itemsets with discount strategies. Inf. Comput. Sci. **11**, 6297–6307 (2014)
17. Liu, Y., Liao, W.-K., Choudhary, A.: A fast high utility itemsets mining algorithm. In: International Workshop on Utility-Based Data Mining, pp. 90–99 (2005)
18. Goethals, B., Zaki, M.J.: The fimi repository (2012)