A Hybrid Algorithm for mining High Utility Itemsets from transaction databases with Discount Notion

Ruchita Bansal
Computer Science
Indraprastha Institute of Information Technology, Delhi (IIIT-D), India

A Thesis Report submitted in partial fulfilment for the degree of

MTech Computer Science

1 July 2015
1. Prof. Vikram Goyal (Thesis Adviser)

2. Prof. Sambuddho Chakravarty (Internal Examiner)

2. Dr. Durvasula Somayajulu (External Examiner)

Day of the defense: 1 July 2015

Signature from Post-Graduate Committee (PGC) Chair:
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 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 thesis, we integrate the notion of discount in state-of-the-art utility-mining algorithms and propose a hybrid-algorithm for efficient mining of 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.
I dedicate my MTech Thesis to My Family and SGI President Dr. Daisaku Ikeda.
Acknowledgements

I would like to express my special appreciation and thanks to my advisor Professor Dr. Vikram Goyal, you have been a tremendous mentor for me. I would like to thank you for encouraging my research and helped me in all the time of research and writing of this thesis. Your advice on both research as well as on my career have been priceless.

Besides my advisor, I would also like to thank Siddharth Dawar (Phd-Data Engineering) for supporting me all the time. I also want to thank you for letting my defense be an enjoyable moment, and for your brilliant comments and suggestions, thanks to you.

I would especially like to thank all faculty members of IIIT-Delhi, especially to Data Engineering Department for imparting the best of knowledge and moulding my future. I am also thankful to all my friends specially Kritika Anand for being with me at each step whenever I needed their support.

A special thanks to my family. Words cannot express how grateful I am to my mother, and father for all of the sacrifices that you’ve made on my behalf. Your prayer for me was what sustained me thus far. I would also like to thank all of my friends who supported me in writing, and incented me to strive towards my goal.
Declaration

This is to certify that the MTech Thesis Report titled *A Hybrid Algorithm for mining High Utility Itemsets from transaction databases with Discount Notion* submitted by Ruchita Bansal for the partial fulfillment of the requirements for the degree of MTech in *Computer Science* is a record of the bonafide work carried out by her under my guidance and supervision at Indraprastha Institute of Information Technology, Delhi. This work has not been submitted anywhere else for the reward of any other degree.

Professor Vikram Goyal
Indraprastha Institute of Information Technology, New Delhi
Contents

List of Figures viii
List of Tables ix

1 Research Motivation and Aim 1
2 Related Work 4
3 Background 6
   3.0.1 Preliminary 6
4 Existing Algorithm 9
   4.0.2 UP-Hist tree 9
   4.0.3 Three-phase algorithm for mining high-utility itemsets with discount strategies 12
4.1 Integration of Discount Notion in State-of-the-art algorithms 12
   4.1.1 UP-Hist Discount algorithm 13
   4.1.2 FHM Discount algorithm 14
5 Research Framework and Solution Approach 17
   5.1 Mining High-utility Itemsets 17
      5.1.1 Example to show the efficiency of our proposed Algorithm 19
6 Experiments and Results 21
7 Conclusion and Future Work 24
References 25
List of Figures

1.1 Effect of Pattern size on the number of High-utility itemsets and Utility-list size on Mushroom Dataset ........................................... 3

4.1 Global UP-Hist Tree ................................................................. 11
4.2 Utility-list of item \{A\} and \{B\} .................................................... 12

6.1 Performance Evaluation on Sparse Datasets ............................... 22
6.2 Performance Evaluation on Dense Datasets ............................... 23
6.3 Scalability on Accidents Dataset ............................................... 23
List of Tables

3.1 Example Database ........................................ 7
3.2 Profit Table .............................................. 7
6.1 Characteristics of Real Datasets .......................... 21
Research Motivation and Aim

High-utility itemset mining finds patterns from a database which have their utility value no less than a minimum utility threshold. In conventional utility mining, the utility function is generally defined as the product of the quantity and profit information associated with each item in the database. However, the notion of utility can be defined according to a given application domain. For example, consider the problem of mining a knowledge-base to be used for the purpose of query keywords recommendation to a user on a documents collection. The documents can be modelled as transactions consisting of words as its items, frequency of a word as an item’s quantity. The relative importance of each word can be modelled by its inverse-document frequency (IDF) over the corpus of documents. High-utility pattern mining also finds applications in anomaly detection such as identifying fraudulent credit card transactions, network intrusions etc., medicine [1], molecular biology [2]. Utility-mining has also been integrated with other concepts like sequential pattern mining [3], episode pattern mining [4] and stream mining [5].

The algorithms on high-utility itemset mining can be classified into two different paradigms, tree-based algorithms [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 compressed representation of the complete transaction database and hence allows mining 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 a Inverted-list like data structure for its working. Liu et al. [7] proposed the utility-list data structure
which is based on Inverted-list and an algorithm which finds \( k \)-length high-utility itemset by intersecting the utility-lists of \( k - 1 \) itemsets. Viger et al. [8] proposed a strategy to improve the performance of HUI-Miner by reducing the number of intersection/join operations. These vertical mining algorithms first generate all the singleton high-utility itemsets and then proceed to 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.

Majority of the utility-mining algorithms consider that the items have constant profits associated them. However, shopkeepers often give discounts on bulk purchase of items to improve their sales and earn better profit. Discount are also given on items which are close to being expired. Even online shopping sites like Flipkart and Snapdeal provide discounts on 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. Li et al. [9] proposed a three-phase algorithm to integrate several discount strategies in the utility mining framework.

In order to avoid the costly join operation for short itemsets in case of Inverted-list based approaches and avoid costly operation of verifying the candidates, we propose a hybrid algorithm which generates high-utility patterns without generating any candidate. Basically, our proposed hybrid 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.

In order to validate our hypothesis that the hybrid algorithm would perform better due to avoidance of costly join operation and candidate verification, we conducted an experiment on a real life dataset named Mushroom available from FIMI repository [10]. The dataset was augmented with quantity and utility values with items as per previous works in the literature. The results of the experiment are shown in Figure 1.1. The
results in the first graph show that the number of short high-utility itemsets is low and the number of high-utility itemsets increases with the length of itemsets. After some point, the number of high-utility itemsets start to decrease as the support of the itemset decreases with the increase in length. The second graph shows that the average size of utility-lists decreases as the length of high-utility itemset increases. It is clear from the first graph that there are very few high-utility patterns of short length. Therefore, tree-based algorithm can be used in the beginning to avoid costly lists intersection/join operation due to long lists being associated with short items, as shown in the second graph.

Our novel research contributions can be summarized as follows:

- We present a unified simple model which integrates the notion of quantity discount into the conventional utility mining framework. Our model can be easily integrated in the state-of-the-art utility-mining algorithms.

- We propose an efficient algorithm namely UP-Hist Hybrid, which has benefits of tree-based as well as Inverted-list based algorithms for mining high-utility itemsets.

- 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 in terms of total execution time.
2

Related Work

Frequent-itemset mining \cite{11, 12, 13} has been studied extensively in the literature. Agrawal et al.\cite{11} proposed an algorithm named Apriori, for mining association rules from market-basket data. Their algorithm was based on the downward closure property \cite{11}. The downward closure property states that every subset of a frequent itemset is also frequent. Park et al.\cite{14} proposed a hash based algorithm for mining association rules which generates less number of candidates compared to Apriori algorithm. Zaki et al.\cite{15} 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. \cite{12} proposed a pattern-growth algorithm to find frequent itemsets by using FP-tree data structure. Vu et al.\cite{16} proposed a hybrid 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 \cite{11}, like we have in frequent-itemset mining scenario.

Several algorithms have also been proposed to find high-utility itemsets. Liu et al.\cite{17} 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.\cite{18} 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+ [19]. The authors proposed effective strategies like DGU, DGN, DLU and DLN to compute 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 utility-list of an itemset stores its exact utility as well as an upper bound on the expected utility values of its supersets by using the remaining utility values stored in the list. The items are sorted and processed in the ascending order of transaction utility. 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. [8] proposed a novel strategy EUCP (Estimated Utility Co-occurrence Pruning) to prune itemsets without performing the join operation. Recently, Li et al. [9] 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. Initially, the algorithm computes the TWU of \{1\}-itemsets in order to find the candidate itemsets of length one. The algorithm then proceeds with the generation of \{2\} candidate itemsets in a level-wise manner. After all the candidate high-utility itemsets are generated, the algorithm computes the exact utility of each candidate by scanning the database.
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.

3.0.1 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 3.0.1. (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. $u(i_p, T_d) = q(i_p, T_d) \times pr(i_p)$.

**Definition 3.0.2. (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 $\sum_{X \subseteq T_d \land i_p \in X} u(i_p, T_d)$.

We also define a utility of a transaction as similar to an itemset over a transaction as given below.
Table 3.1: Example Database

<table>
<thead>
<tr>
<th>TID</th>
<th>Transaction</th>
<th>TU</th>
</tr>
</thead>
<tbody>
<tr>
<td>T₁</td>
<td>(A : 1) (C : 1) (D : 1)</td>
<td>9</td>
</tr>
<tr>
<td>T₂</td>
<td>(A : 2) (C : 6) (E : 2) (G : 5)</td>
<td>47</td>
</tr>
<tr>
<td>T₃</td>
<td>(A : 1) (B : 2) (C : 1) (D : 6) (E : 1) (F : 5)</td>
<td>33</td>
</tr>
<tr>
<td>T₄</td>
<td>(B : 4) (C : 3) (D : 3) (E : 1)</td>
<td>18</td>
</tr>
<tr>
<td>T₅</td>
<td>(B : 2) (C : 2) (E : 1) (G : 2)</td>
<td>23</td>
</tr>
</tbody>
</table>

Table 3.2: Profit Table

<table>
<thead>
<tr>
<th>Item</th>
<th>Profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
</tr>
<tr>
<td>E</td>
<td>5</td>
</tr>
<tr>
<td>F</td>
<td>2</td>
</tr>
<tr>
<td>G</td>
<td>7</td>
</tr>
</tbody>
</table>

Definition 3.0.3. (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 3.1 and the profit associated with each item in Table 3.2. 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 3.1.

Once we have utility of an itemset over a transaction, the utility of an itemset over the whole database can be computed by simply computing the sum of utility of that itemset in each transaction.

Definition 3.0.4. (Utility of an itemset in Database). The utility of an itemset \( X \) in database \( D \) is denoted as \( u(X) \) and defined as \( \sum_{X \subseteq T_d \land T_d \in D} u(X, T_d) \).

For example, \( u(B, C) = u(\{B, C\}, T_2) + u(\{B, C\}, T_3) + u(\{B, C\}, T_4) + u(\{B, C\}, T_5) = 8 + 18 + 10 = 36 \). Once we have the utility value of an itemset over the database, given the value of utility threshold parameter we can define high-utility itemsets as follows.

Definition 3.0.5. (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 \( \text{min\_util} \).

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 \( \text{min\_util} = 40 \), then \( \{C, E\} \) is a high-utility itemset. However, if \( \text{min\_util} = 50 \), then \( \{C, E\} \) is a low-utility itemset.

Problem Statement. Given a transaction database \( D \) and a minimum utility threshold \( \text{min\_util} \), the aim is to find all the itemsets which have high-utility.
The high-utility itemsets at minimum threshold 50 are \{CG\}, \{EG\}, \{ACG\}, \{AEG\}, \{BCE\}, \{CEG\}, \{ACEG\} and \{BCDE\}. We now describe the concept of transaction utility and transaction weighted downward closure (TWDC)\cite{20}.

**Definition 3.0.6. (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 \(\sum_{X \subseteq T \land T \in D} TU(T_d)\).

**Definition 3.0.7. (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, \(TU(T_1) = u(\{ACD\}, T_1) = 9; TWU(\{A\}) = TU(T_1) + TU(T_2) + TU(T_3) = 89\). If \(min_{util} = 80\), \{A\} is a HTWUI. However, if \(min_{util} = 90\), \{A\} and any of its supersets are not HTWUIs.
Existing Algorithm

4.0.2 UP-Hist tree

Each node $N$ in UP-Hist tree [21] consists of a name $N.item$, overestimated utility $N.nu$, support count $N.count$, a histogram, 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 root of the tree is a special empty node which points to its child nodes. The support count of a node $N$ along a path is the number of transactions contained in that path that have the item $N.item$. In order to facilitate efficient traversal, a header table is also maintained. The header table has three columns, Item, TWU and Link. The nodes in a UP-Tree along a path are maintained in descending order of their TWU values. 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.

Algorithm 1: UP-Hist Growth($T_\times, H_\times, X$)

**Data:** A UP-Hist tree $T_\times$, a header table $H_\times$ for $T_\times$, an itemset $X$, a minimum utility threshold $\text{min}_\text{util}$

**Result:** All candidate high-utility itemsets in $T_\times$

1. **foreach** each entry $\{a, i\}$ in $H_\times$ do
   2. Compute $\text{node.nu}$ by following the links from the header table for each item $\{a, i\}$.
   3. Also, compute the upper bound utility of item $\{a, i\}$ denoted by $UB(\{a, i\})$ and $LB(\{a, i\})$ respectively.
   4. if $\text{node.nu}(a, i) \geq \text{min}_\text{util}$ then
      5. if $\text{LB}_{\text{sum}}(a, i) \geq \text{min}_\text{util}$ then
         6. Add itemset $\{a, i\}$ to the set of Verified High utility itemsets.
         7. Construct the CPB of $Y = X \cup a, i$
      8. else if $UB_{\text{sum}}(a, i) \geq \text{min}_\text{util}$ then
         9. Construct PHUI $Y = X \cup a, i$ and CPB of Y.
     10. end
     11. else
         12. Construct the CPB of $Y = X \cup a, i$ only
     13. end
   14. end
   15. Put local promising items in $Y - \text{CPB}$ into $H_Y$ and apply DLU to reduce path utilities.
   16. Insert every reorganized path into $T_Y$ after applying DLN.
   17. if $T_Y \neq \text{null}$ then
      18. Call UP-Hist Growth($T_Y, H_Y, Y$)
   19. end
20. end
21. end

**Definition 4.0.8. (Histogram).** A histogram $h$ is a set of pairs $(q_i, num_i)$, where $q_i$ is an item quantity and $num_i$ is the number of transactions that contain $q_i$ copies of an item.

The global UP-Hist tree for the utility threshold value 36 and transaction database shown in table 3.1 is shown in Figure 4.1. UP-Hist Growth is a two-phase recursive algorithm which generates candidate high-utility itemsets in the first phase. The algorithm basically starts with an empty pattern and extends that while exploring the UP-Hist tree from bottom-most node to the root node. Each recursive call results into increasing the pattern length by one and have its own local UP-Hist tree for processing. The histogram associated with each node helps in computing the minimum and
Definition 4.0.9. (minC). Let \( h \) be a histogram, associated with an item-node \( N_i \), consisting of \( n \) 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 4.0.10. (maxC). Let \( h \) be a histogram, associated with an item-node \( N_i \), consisting of \( n \) 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 \( h = \{ < 1, 2 >, < 2, 1 >, < 3, 1 >, < 6, 1 > \} \). \( minC(C, 3) \) and \( maxC(C, 3) \) is 6 and 11 respectively. These quantity estimates are used by the algorithm to finally compute better estimates for lower-bound and upper-bound value of any itemset. The readers can refer to [21] for more details of the algorithm.

FHM [8] 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 Figure 4.2. FHM works similar to Apriori algorithm. However it first computes high TWU singleton itemsets in the first scan of the database and
### 4.1 Integration of Discount Notion in State-of-the-art algorithms

#### Figure 4.2: Utility-list of item \{A\} and \{B\}

<table>
<thead>
<tr>
<th>TID</th>
<th>Iutils</th>
<th>Rutils</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>22</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TID</th>
<th>Iutils</th>
<th>Rutils</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>23</td>
</tr>
</tbody>
</table>

the utility-list of singleton itemsets with high \(TWU\) is constructed in the next scan of the database. As like Apriori, 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.

#### 4.0.3 Three-phase algorithm for mining high-utility itemsets with discount strategies

Recently, Li et al. [9] 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 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 generation of many candidate itemsets. In the third phase, exact utility of candidate-itemsets is computed to find the actual high-utility itemsets.

### 4.1 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 (HUI-Miner).
4.1 Integration of Discount Notion in State-of-the-art algorithms

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

Definition 4.1.1. (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

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

$min\_pr(x)$ is the minimum profit per unit item of $x$ in database $D$ after applying discount rules if any for item $x$.

Definition 4.1.2. (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

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

$max\_pr(x)$ is the maximum profit per unit item of $x$ in database $D$ without applying any discount rule.

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 4.1.3. (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

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

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

13
4.1 Integration of Discount Notion in State-of-the-art algorithms

Definition 4.1.4. (Lower-bound utility) Given an itemset \( I = \langle a_1, a_2, ..., a_k \rangle \) 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

\[
lb(I) = \sum_{i=1}^{k} \max(minC(a_i, s) \times \min lpr(a_i), MinU(a_i, s))
\]

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

The utility values of an itemset \( I \) are correct lower bound and upper bound estimates of the exact utility of \( I \).

4.1.2 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 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
4.1 Integration of Discount Notion in State-of-the-art algorithms

mining-based approach like FHM.

**Algorithm 2: The Construct Procedure**

**Data:** P: an itemset, Px: the extension of P with an item x, Py: the extension of P with an item y

**Result:** the utility-list of Pxy

1. UtilityListOfPxy ← ∅;
2. foreach tuple ex ∈ Px:utilitylist do
3.    if ∃ ey ∈ Py:utilitylist and ex:tid = ey:tid then
4.        if P:utilitylist ≠ ∅ then
5.            Search element e ∈ P:utilitylist such that e:tid = ex:tid.;
6.            exy ← (ex:tid; ex:iutil + ey:iutil; e:iutil; ey:rutil);
7.        end
8.    else
9.        exy ← (ex:tid; ex:iutil + ey:iutil; ey:rutil)
10.    end
11. UtilityListOfPxy ← UtilityListOfPxy ∪ exy;
12. end
13. return UtilityListPxy;

**Algorithm 3: FHM algorithm**

**Data:** D: a transaction database, minutil: a user-specified threshold

**Result:** All candidate high-utility itemsets in $T^*_x$

1. Scan D to calculate the TWU of single items;
2. $I^*$ ← each item i such that $TWU(i) \leq minutil$;
3. Let the total order of TWU ascending values on $I^*$;
4. Scan D to built the utility-list of each item $i \in I^*$ and build the EUCS structure;
5. Search ($\emptyset$, $I^*$, minutil, EUCS);
4.1 Integration of Discount Notion in State-of-the-art algorithms

**Algorithm 4: The Search Procedure**

**Data:** P: an itemset, ExtensionsOfP: a set of extensions of P, the minutil threshold, the EUCS structure

**Result:** All candidate high-utility itemsets in $T_x$

1. **foreach** itemset $Px \in$ ExtensionsOfP do
2.     if $\sum(\text{utility of } Px : \text{utility list : iutils}) \geq \text{min_util}$ then
3.         output $Px$;
4.     end
5.     if $\sum(\text{utility of } Px : \text{utility list : iutils}) + \sum(\text{utility of } Px : \text{utility list : rutils}) \geq \text{min_util}$ then
6.         ExtensionsOfPx $\leftarrow \emptyset$; **foreach** itemset $Py \in$ ExtensionsOfP such that $y \succ x$ do
7.             if $\exists (x, y, c) \in \text{EUCS}$ such that $c \geq \text{min_util}$ then
8.                 $Pxy \leftarrow Px \cup Py$;
9.                 $Pxy: \text{utility list} \leftarrow \text{Construct (P, Px, Py)}$;
10.                ExtensionsOfPx $\leftarrow \text{ExtensionsOfPx} \cup Pxy$;
11.             end
12.         end
13.     end
14. end
15. end
Research Framework and Solution Approach

5.1 Mining High-utility Itemsets

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

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 UP-Hist algorithm is called recursively.

We would like to highlight that there is an issue in combining UP-Hist and FHM directly. The FHM algorithm sorts the items in a transaction in increasing $TWU$ value and then construct the utility-list for efficiency reasons. Therefore to keep both the efficiency as well as correctness, the ordering of items should remain intact while integrating UP-hist growth and FHM. In our case, we create a utility list for an arbitrary itemset when its estimate utility becomes larger than or equal to the threshold. However, a naive approach to create a utility-list would incur very high costs. Therefore, we use a memoization technique to make the utility-list construction efficient. While constructing the utility-list of an itemset, we store utility-list of each prefix itemset in a hash table in order to reuse them later. In future, when the construction of the utility-list of any itemset $I$ is done, we first check if there exists any best match for the current itemset that can be used.

initially algorithm is calling by these parameters i.e UP-Hist Hybrid discount Growth($T_x, H_x, null$) where $T_x$ is specified global UP-Hist Tree data structure, $H_x$ is a header table which is associated with the specified data structure and itemset initially null at that time.
5.1 Mining High-utility Itemsets

Algorithm 5: UP-Hist Hybrid Discount Growth($T_x, H_x, X$)

Data: A UP-Hist tree $T_x$, a header table $H_x$ for $T_x$, an itemset $X$, a minimum utility threshold $\text{min\_util}$

Result: All candidate high-utility itemsets in $T_x$

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

5.1.1 Example to show the efficiency of our proposed Algorithm

We now present an example to illustrate the working of our proposed algorithm. We consider the example database shown in Table 3.1 with a threshold utility of 36. The global UP-Hist tree is shown in Figure 4.1. 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 are called Conditional Pattern Base (CPB). The TWU of each items 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 25.33, 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 sum of node utility is 41, which is greater than the minimum threshold. The upper-bound utility value of itemset \( \{EBD\} \) is 40.33, which is greater than the threshold. Therefore, the algorithm now constructs a utility-list for itemset \( \{EBD\} \) and switches to FHM strategy.
Experiments and Results

In this section, we compare the performance of our proposed hybrid algorithm against the state-of-the-art algorithms UP-Hist Growth \[21\] and FHM \[8\]. We integrated our model in FHM and UP-Hist Growth to make them comparable with our proposed algorithm. We also implemented the three-phase algorithm integrating discount strategies \[9\]. 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 \ref{tab:real-datasets}. 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 \[10\]. 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

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#T_x</th>
<th>Avg. length</th>
<th>#Items</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kosarak</td>
<td>9,90,002</td>
<td>8.1</td>
<td>41270</td>
<td>Sparse</td>
</tr>
<tr>
<td>Retail</td>
<td>88,162</td>
<td>10.3</td>
<td>16470</td>
<td>Sparse</td>
</tr>
<tr>
<td>Accidents</td>
<td>3,40,183</td>
<td>33.8</td>
<td>468</td>
<td>Dense</td>
</tr>
<tr>
<td>Connect</td>
<td>67,557</td>
<td>43</td>
<td>129</td>
<td>Dense</td>
</tr>
</tbody>
</table>
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 is shown in Figure 6.1. The results show that our hybrid 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 is shown in Figure 6.2. 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 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 hours. We also conduct experiments to evaluate the scalability of our algorithm on Accidents dataset and the result is shown in Figure 6.3. The result shows that performance of our algorithm improves with increase in the number of transactions.

Figure 6.1: Performance Evaluation on Sparse Datasets
Figure 6.2: Performance Evaluation on Dense Datasets

Figure 6.3: Scalability on Accidents Dataset
Conclusion and Future Work

In this Thesis, we introduced a simple unified model to introduce the notion of discount while extracting high-utility itemsets from transaction databases. We also proposed modified version of UP-Hist data structure and a novel algorithm UP-Hist Hybrid discount Growth, for mining high-utility itemsets. We further proposed techniques to compute tight lower and upper bounds on the estimated utility, which helps in pruning the search space. We compared the performance of our algorithm using different data structures and the results show that our proposed algorithm outperforms the state-of-the-art algorithms in terms of number of candidates and execution time. The results show that UP-Hist data structure is better compared to other data structures for dense as well as sparse datasets.
References


[12] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In ACM SIGMOD, 29, pages 1–12. ACM, 2000. 4


