UP-Hist Tree: Efficient Data Structure for High Utility Pattern Mining from Transaction Databases

Vikram Goyal, Siddharth Dawar

Indraprastha Institute of Information Technology, Delhi

Abstract. High-utility itemset mining is an emerging research area in the field of Data Mining. Several algorithms were proposed to find high-utility itemsets from transaction databases and use a data structure called UP-tree for their working. However, algorithms based on UP-tree generate a lot of candidates due to limited information availability in UP-tree for computing utility value estimates of itemsets. In this paper, we present a data structure named UP-Hist tree which maintains a histogram of item quantities with each node of the tree. The histogram allows computation of better utility estimates for effective pruning of the search space. Extensive experiments on real as well as synthetic datasets show that our algorithm based on UP-Hist tree outperforms the state of the art algorithms in terms of the total number of candidate high utility itemsets generated as well as total execution time. The UP-Hist tree takes low memory ranging from few KB's to MB's only.

Keywords: Frequent Itemset Mining, Utility Mining, Pattern Mining, Data Mining

1 Introduction

High-utility pattern mining[1–4] finds patterns from a database that have their utility value no less than a given minimum utility threshold. The utility of a pattern defines its importance and makes mined patterns more relevant for certain applications. Primarily, the interest in utility patterns arises as it allows to associate relative importance to different items, and accounts for multiplicity of items. On the other hand, frequent-pattern mining can't be used to find high utility patterns, due to its limitation of treating every item with equal importance with no use of item-quantity information. Applications like retail stores, where each item has different profit values and a transaction can have multiple copies of an item, will have a direct role of high utility pattern mining. In this scenario, the patterns can be interpreted as itemsets that contribute to the majority of the profit, and can be used for deciding inventory of a retail store. Similar to retail stores, utility mining also finds its applications in web click stream analysis[5], bio-medical data analysis[6] and mobile E-commerce environment[7].

The majority of the algorithms on utility pattern mining is of type pattern growth and use an efficient data structure named UP-tree for their working.

However, these algorithms generate a large number of candidate patterns in the first phase, which are then verified for their high utility property in the second phase. To reduce the number of candidate patterns generated in the first phase in pattern growth algorithms, there are primarily two steps: i) identifying better estimates of the utility value of patterns and (ii) systematic search of space for patterns, using the estimates. The utility-value estimates allow early pruning of non-candidate itemsets and avoids exploration of unpromising nodes in the tree. We believe that any data structure which helps into computation of better estimates will improve the performance of the mining algorithms by effectively pruning the search space. Along the similar lines of thought, authors in [2] associated with each tree node a minimum quantity information and have observed improvement in performance.

In this paper, we propose a data structure named as UP-Hist tree that supports computing better estimates as compared to the previous approaches. In UP-Hist tree, we associate a histogram which represents frequency distribution of item-quantities with each node of the UP-tree and use it for storing the quantity information of transactions. We observe that use of UP-Hist tree improves the performance of utility-pattern mining algorithm. However, histograms require some extra storage as compared to the previous approaches, but still the space required is a couple of MB's which is not a big requirement in the era of cloud-computing and big-data. We propose an algorithm called UP-Hist Growth, which uses the UP-Hist tree data structure to mine high-utility patterns. The algorithm is similar to UP-Growth, but effectively uses the histograms available with tree nodes.

Our contributions can be summarized as follows:

- We propose a novel data structure UP-Hist tree and an algorithm UP-Hist Growth for discovering high utility itemsets from a transaction database.
- We prove that the estimates computed by UP-Hist tree are correct and better than the state-of-the-art approaches.
- We conduct extensive experiments on real as well as synthetic datasets to demonstrate the performance of our proposed solution. The results confirm that our proposed solution is scalable and outperforms state-of-the-art algorithms in terms of the number of candidates and execution time.

2 Related Work

Frequent-itemset mining [8–10] has been studied extensively in the literature and several algorithms have been proposed. 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. The downward closure property states that every subset of a frequent itemset is also frequent.

High-utility itemset mining[1] finds itemsets from the database which have their utility no less than a minimum utility threshold. Several algorithms have been proposed to find high utility itemsets. Liu et al.[3] proposed a two-phase algorithm for mining high utility itemsets. The candidate high utility itemsets were generated in the first phase and verification was done in the second phase. Ahmed et al.[11] proposed a data structure called IHUP-tree and another two-phase algorithm to mine high utility patterns incrementally from dynamic databases. However, the above algorithms generate a lot of candidate itemsets in the first phase. In order to reduce the number of candidates, Tseng et al.[12] proposed a new data structure called UP-tree and proposed algorithms namely, UP-Growth [12] and UP-Growth+ [2]. The authors proposed effective strategies like DGU, DGN, DLU and DLN to reduce the overestimated utilities. However, we observed that a histogram can be augmented with every node of the UP-tree and can be used to reduce the overestimated utilities further. We present strategies to compute utility estimates using the histograms and demonstrate the better performance of our proposed strategy.

Several algorithms have been proposed to find top-k utility itemsets in transactions [13], sequence databases [14] and data streams [15]. Most top-k algorithms use a common search strategy for finding the results. The idea is to use a buffer of size k and store the intermediate results in this buffer. The process of finding top-k utility itemsets is repeated until it is guaranteed that no more itemsets can become a part of the result. Wu et al. [13] proposed an algorithm named TKU based on UP-Growth [12] for finding top-k high utility itemsets. We believe that these works can also get the benefit of detailed information available in the form of histogram to improve their performance.

3   Background

In subsection 3.1, we give some definitions given in earlier works and describe the problem statement formally. We also briefly discuss the UP-tree data structure in subsection 3.2.

3.1   Preliminary

We have a set of m items $I = \{I_1, I_2, ..., I_m\}$, where each item has a profit $pr(i_p)$ associated with it. An itemset X of length k is a set of k distinct items $X = \{I_1, I_2, ..., I_k\}$, where for $i \in 1.....k$, $I_i \in I$. 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.

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

Table 1: *Example Database*

TID	Transaction	TU
T_1	$(A:1)\,(C:10)\,(D:1)$	17
T_2	$(A:2)\,(C:6)\,(E:2)\,(G:5)$	27
T_3	$(A:2)\,(B:2)\,(D:6)\,(E:2)\,(F:1)$	37
T_4	$(B:4)\,(C:13)\,(D:3)\,(E:1)$	30
T_5	$(B:2)\,(C:4)\,(E:1)\,(G:2)$	13
T_6	$(A:6)\,(B:1)\,(C:1)\,(D:4)\,(H:2)$	43

Table 2: *Profit Table*

Item	A	B	C	D	E	F	G	H
Profit	5	2	1	2	3	5	1	1

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

Let us consider the example database shown in Table 1 and the profit values in Table 2. The utility of item $\{A\}$ in $T_3 = 2 \times 5 = 10$ and the utility of itemset $\{A, B\}$ in T_3 denoted by $u(\{A, B\}, T_3) = u(A, T_3) + u(B, T_3) = 10 + 4 = 14$.

Definition 3. *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, $u(A, B) = u(\{A, B\}, T_3) + u(\{A, B\}, T_6) = 14 + 31 = 55$.

Definition 4. *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, $u(A, C) = u(\{A, C\}, T_1) + u(\{A, C\}, T_2) + u(\{A, C\}, T_6) = 15 + 16 + 31 = 62$. If $min\_util = 50$, then $\{A, C\}$ is a high utility itemset. However, if $min\_util = 75$, then $\{A, C\}$ is a low utility itemset.

Problem Statement. Given a transaction database D and a minimum utility threshold min_util, the aim is to find all the high utility itemsets i.e. itemsets which have utility no less than min_util.

We will now describe the concept of transaction utility and transaction weighted downward closure(TWDC)[1].

Definition 5. *The transaction utility of a transaction T_d is denoted by $TU(T_d)$ and defined as $u(T_d, T_d)$.*

For example, the transaction utility of every transaction is shown in Table 1.

Definition 6. *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 7. *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, $TU(T_1) = u(\{ACD\}, T_1) = 17; TWU(\{A\}) = TU(T_1) + TU(T_2) + TU(T_3) + TU(T_6) = 124$. If $min\_util = 60$, $\{A\}$ is a HTWUI. However, if $min\_util = 130$, $\{A\}$ and any of its supersets are not HTWUIs.

3.2   UP-Tree

Each node N in UP-tree [12] consists of a name $N.item$, overestimated utility $N.nu$, support count $N.count$, 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 $N.item$. $N.nu$ is the overestimated utility of an itemset along the path from node N to the root. 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.

4   Mining High Utility Itemsets

In this section, we will present the construction process of UP-Hist tree. Subsequently, we will show how UP-Hist tree can be used to compute better estimates of utility value for an itemset as well as a node($N.nu$). We also prove that the utility estimates computed by our method are correct and better as compared to the state-of-art approaches, namely UP-Growth [12], UP-Growth+ [2].

4.1   Construction of UP-Hist Tree

Primarily, in the UP-Hist tree we augment a histogram with every node of the UP-tree. A histogram at a node stores the quantity information for a specific set of transaction that contributes to the node utility.

Definition 8. *A histogram h for an item-node N_i 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 process of constructing UP-Hist tree is similar to the previous approaches [2], [12] and requires two scans of the database. In the first scan, items are ordered on the basis of their TWU values and non-candidates are discarded. A header table is also created that explicitly stores the TWU values with each item and maintains items in non-increasing order of their TWU value. During the second scan of the database, each transaction is first reorganized and then inserted into the UP-Hist tree. Reorganization of a transaction reduces the TU value by the utility value of the discarded items as well as reorders the remaining items in the
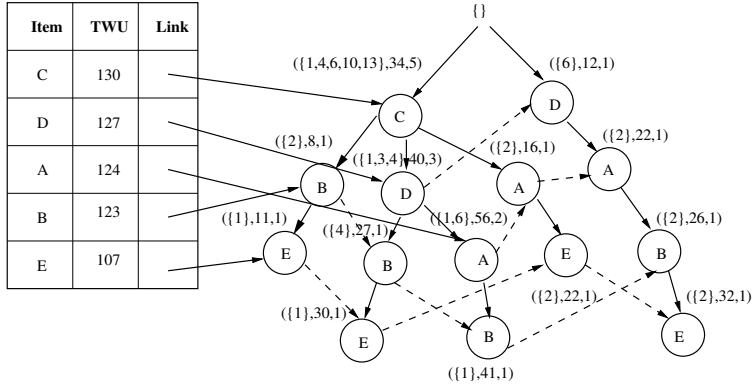
Item	TWU	Link
C	130	
D	127	
A	124	
B	123	
E	107	

Tree nodes: {} root. ({1,4,6,10,13},34,5) → C; ({6},12,1) → D. ({2},8,1), ({1,3,4},40,3) → B, D; ({2},16,1), ({2},22,1) → A, A. ({1},11,1), ({4},27,1), ({1,6},56,2) → E, B, A; ({2},26,1) → B. ({1},30,1), ({2},22,1), ({2},32,1) → E, B, E. ({1},41,1).

Fig. 1: Global UP-Hist tree

transaction using their RTU values. Similar to the basic UP-tree, tree traversal of UP-Hist tree is supported by links maintained in the header table and at each node of the tree.

The process of inserting reorganized transactions in the UP-Hist tree is as follows: each transaction is processed from the beginning and matched with nodes in the tree starting from the root. In case the item of a transaction matches with the node's item, the histogram associated with that node is updated with the item's quantity value, i.e. if a pair p with the same quantity value exists then count is incremented by one, otherwise a new pair is added to the histogram. The node utility value is also updated by the utility-value of the transaction-prefix and the support is incremented by one. In the second case, when the item of transaction does not match with any node at that level, a new node is created for that item and an empty histogram is associated with the node. Then, a pair $(quantity, support)$ is added in the histogram with its first component value set as the item's quantity and the support (t_c) is set to one. For the example database given in Table 1 and profit Table 2, the global UP-Hist tree will be as shown in Figure 1. The histograms in the figure are shown as the list of item-quantity values for the reason of legibility, instead of a list of pairs.

4.2 Generating High-Utility Itemsets from UP-Hist Tree

In this subsection, we present our approach to calculate better utility estimates of itemsets and nodes using histograms. Subsequently, we give a proof of correctness of our estimates. Finally, we illustrate the advantage of using histograms with an example.

Our algorithm 1 is a pattern-growth recursive algorithm. The algorithm picks every item in a bottom-up manner from the header. If the picked item is of high utility and can generate high utility itemsets, a local tree is generated for that item, which is further explored in a recursive manner. At every expansion of

Algorithm 1 UP-Hist Growth(T_x, H_x, X)

1: **for** entry i_k in H_x **do**
2:     **if** $nu_{sum}X \geq min\_util$ **then**
3:         Consider $Y = X \cup i_k$ as a candidate and construct CPB of Y.
4:         Put local promising items in $Y - CPB$ into H_Y and create local UP-Hist tree.
5:         **if** $T_Y \neq null$ **then** Call UP-Hist Growth(T_Y, H_y, Y)
6:         **end if**
7:     **end if**
8: **end for**

---

a prefix, the utility of prefix in the local tree is estimated to decide whether further exploration is worthy. Our algorithm generates these utility estimates using histograms. We discuss the strategies of computing estimates and process of constructing a local UP-Hist tree further.

In order to compute the estimates for an item-node N_i of a tree having a support count s, we define two primitive operations, namely $minC(N_i, s)$ and $maxC(N_i, s)$, that computes the minimum (lb) and maximum (ub) number of item copies for a given number of transactions.

Definition 9. *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 maximal number fulfilling $k \leq \sum_1^k num_i$.*

Definition 10. *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 $k \leq \sum_1^k num_i$.*

Consider the histogram at node C in Figure 1, $h = \{< 1, 1 >, < 4, 1 >, < 6, 1 >, < 10, 1 >, < 13, 1 >\}$. The $minC(C, 3)$ and $maxC(C, 3)$ will be 11 and 29 respectively.

In the process of pattern growth, any intermediate pattern is checked whether the pattern is high-utility or not. It requires to estimate the utility of the pattern and is computed using $minC(.)$ and $maxC(.)$, as given further.

Definition 11. *Given an itemset $I = < a_1, a_2, ..., a_k >$ corresponding to a path in a UP-Hist tree , with support count value as s, the ub and lb utility values of I are computed as follows: $ub(I) = \sum_{i=1}^{k} maxC(a_i, s) * pr(a_i)$. $lb(I) = \sum_{i=1}^{k} minC(a_i, s) * pr(a_i)$.*

We also use histograms to remove unpromising items during the construction of local UP-Hist tree. The local tree construction is basically a two-step process where, first a conditional pattern base (CPB) of a prefix itemset is created. The conditional pattern base (CPB) of a prefix itemset is the collection of paths

from which the prefix itemset is reachable from the root of the local tree. In our case, each path in the CPB will have a path utility value and each item in a pattern will have a partial histogram associated with it. The partial histogram is computed using the support count of a path as follows:

Definition 12. *Given a path p with a support count s, a partial histogram for an item-node N_i consists of the entries of N_i's histogram used to compute $minC(N_i,s)$ and $maxC(N_i,s)$ score.*

In the second step local-tree construction, the $CPB - paths$ are reorganized by removing the unpromising items to produce reorganized paths, and the utility of a reorganized path is defined as follows.

Definition 13. *Reorganized path utility of a path p, with a support count s due to removal of a set of unpromising local nodes R, is computed as follows: $p.nu_{new} = p.nu - \sum_{n \in R} minC(n,s) * pr(n)$, where $p.nu$ is old path-utility before reorganization.*

The obtained CPB consisting of reorganized paths is then used to create a local UP-Hist tree similar to the process of creating a local UP-tree. However, we merge two histograms in the process using standard bag-union operation. After the paths are reorganized, the new utility of every node N_i along that path is calculated as shown below:

$$(N_i.nu)_{new} = (N_i.nu)_{old} + p.nu - \sum_{n \in R} minC(n,s) * pr(n).$$

Claim 1. *The utility values of an itemset I i.e $lb(I)$ and $ub(I)$ are correct lower and upper bound estimates of the exact utility of I*

Proof. As per Definition 11, the lower bound utility estimate of I i.e. $lb(I)$ is computed as a summation of the product of $minC(a_i,s)$ and profit $pr(a_i)$, for each item $a_i \in I$. The exact utility of the itemset I is computed as summation of product of the exact number of copies of each item $a_i \in I$ and the profit $pr(a_i)$ associated with each item. It is trivial to see the way $minC(.)$ is computed as per Definition 9, the actual quantity of each item a_i in I can't be less than computed by $minC(.)$. Similar argument holds for $ub(I)$, which proves the claim. $\qquad\square$

Claim 2. *The estimated reorganized path utility($p.nu_{new}$) and the new utility of every node($(N_i.nu)_{new}$) computed by UP-Hist Growth is better compared to UP-Growth, UP-Growth+.*

Proof. The reorganized path and node utilities are computed by removing the utilities of unpromising items. The authors [2] used the minimum item utility of an item i denoted by $miu(i)$ and minimum node utility of a node N, denoted as $N.mnu$ to compute the estimates. $miu(i)$ is the minimum quantity of item i in the database and can be represented as $global\_hist(1)$ i.e. the lowest quantity value in the global histogram. $N.mnu$ is the minimum quantity of item $N.name$ in the subset of transactions covered in the path p containing item $N.name$ in the tree and can be represented as $hist(1)$, where $hist(1)$ is the lowest item from

the global or the local histogram of item i. It is trivial to observe,
$\sum_{n \in R} minC(n,s) * pr(n) \geq \sum_{n \in R} s * n.mnu * pr(n) \geq \sum_{n \in R} s * miu(n) * pr(n)$
which proves the claim.

$\square$

Next, we present an example to show the effectiveness of our utility estimates. Let us consider the example database as shown in Table 1 and let $min\_util = 75$. In the first pass of the database, transaction weighted utilization (TWU) of every distinct item is calculated. $\{F\}, \{G\}$ and $\{H\}$ are the low utility items as their TWU is below the minimum utility threshold. The transactions are then reorganized by removing the unpromising (low utility) items and sorting the items within a transaction in decreasing order of their TWUs. Every reorganized transaction is inserted one by one to create a global UP-Hist tree as shown in Figure 1. Let us now process the local tree created by processing item $\{A\}$ from the header table. Item $\{A\}$ is a candidate high utility itemset, since its reorganized transaction utility is 94, which is greater than the minimum utility threshold. The conditional pattern base of $(\{A\} - CPB)$ is created and items in the CPB are processed. $(\{A\} - CPB)$ consists of paths $< CD >, < C >$ and $< D >$ with path utility 56, 16 and 22. The transaction utility of items $\{C\}$ and $\{D\}$ is 72 and 78. Therefore, $\{C\}$ is an unpromising item and its utility must be subtracted to get the reorganized path utilities. The reorganized utility of the path $< CD >$ by UP-Growth is computed as shown below.
$p.nu_{new}(< CD >, A - CPB) = 56 - miu(C) \times s(c) = 56 - 1 \times 2 = 54$.

$p.nu_{new}(< CD >, \{A\} - CPB)$ computed by UP-Growth+ is same as $c.mnu$ is also 1. The estimated utility of the itemset $< AD >$ by UP-Growth and UP-Growth+ is equal to the sum of path utility of $< CD >$ and $< D >$ in $\{A\} - CPB$ i.e. 76.

Now, we will calculate the estimated utility using our histogram. The support of the unpromising item C is 2 and $minC(C,2)$ is 5. The path utility of path $< CD >$ using the histogram of item $\{C\}$ $\{< 1,1 >, < 4,1 >, < 6,1 >, < 10,1 >, < 13,1 >\}$ is given below:

$pu(< CD >, \{A\} - CPB) = 56 - minC(C,2) * pr(C) = 56 - 5 = 51$.

The estimated utility of itemset $< AD >$ is 73. Therefore, $< AD >$ is a potential high utility itemset according to the UP-Growth and UP-Growth+ algorithm, but a low utility itemset according to our algorithm.

5 Experiments and Results

In this section, we compare the performance of our proposed algorithm UP-Hist Growth against the state-of-the-art approaches, i.e. UP-Growth [12], UP-Growth+[2], on various real and synthetic datasets. The description of the various datasets is shown in Table 3. We implemented all the algorithms in Java on Eclipse 3.5.2 platform with JDK 1.6.0_24. The experiments were performed on an Intel Xeon(R) CPU=26500@2.00 GHz with 64 GB RAM. The datasets Accidents, Mushroom and Chess were obtained from FIMI Repository [16]. The
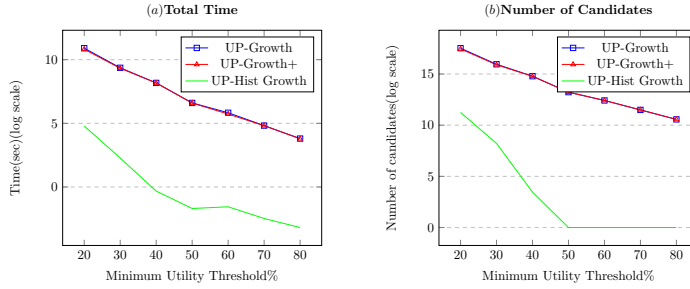
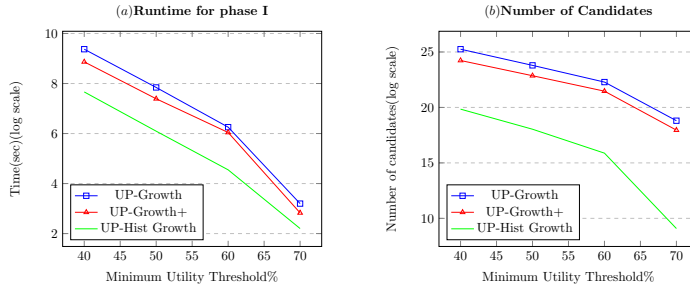Fig. 2: Performance Evaluation on Mushroom Dataset

Fig. 3: Performance evaluation on Chess Dataset

Foodmart dataset was acquired from Microsoft Foodmart 2000 database. Only the Foodmart dataset contained the quantity and external utility values for each item. The quantity and external utility for other datasets were generated using log normal distribution. We compared the algorithms on the basis of number of candidates generated in the first phase and total execution time. We report execution time by taking the average of 20 iterations of each result to minimize the effect of other system parameters. The results for the dense datasets Mushroom and Chess are shown in Figure 2 and 3 respectively. The graphs show that our proposed approach outperforms the state-of-the-art approaches in terms of total execution time and the number of candidates. The reduction in the number of candidates reduces the number of local trees generated during the mining process. We also evaluated the performance of our algorithm on the sparse Foodmart dataset and the results, as shown in Figure 4, show that our algorithm performs better on sparse as well as dense databases. We also performed experiments to show the scalability of our proposed algorithm. The experiments were performed on the Accidents dataset and the results are shown in Figure 5. The results demonstrate that the number of candidates and execution time of every algorithm increases with an increase in the number of transactions. However, our proposed algorithm UP-Hist Growth, still generates the least number candidates compared to the other state-of-the-art algorithms.
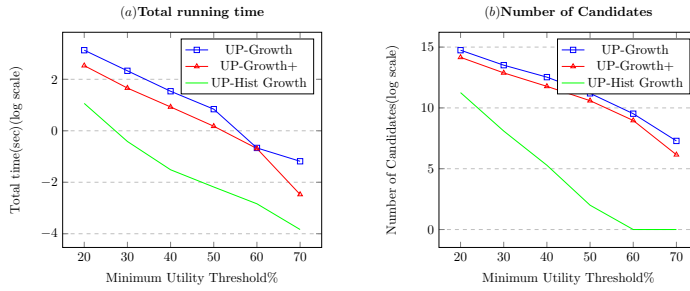
Fig. 4: Performance Evaluation on Foodmart dataset

Table 3: *Characteristics of Datasets*

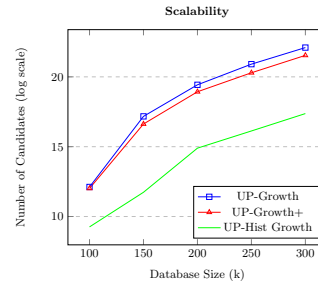| Dataset	#T_x	Avg. length	#Items	Type
Accidents	340,183	33.8	468	Dense
Chess	3,196	37.0	75	Dense
Foodmart	227	17.88	1559	Sparse
Mushroom	8,124	23.0	119	Dense

Fig. 5: Effect of Scalability

6   Conclusion

In this paper, we proposed a novel data structure, UP-Hist tree for finding high utility itemsets. The inclusion of histogram reduced the estimated utility and helped in pruning the search space better. Experimental results on real and synthetic datasets demonstrate the better performance of our proposed data structure compared to the state-of-the-art algorithms in terms of total execution time and number of candidates.

References

1. Ying Liu, Wei-keng Liao, and Alok Choudhary. A fast high utility itemsets mining algorithm. In *International workshop on Utility-based data mining*, pages 90–99. ACM, 2005.
2. Vincent S Tseng, Bai-En Shie, Cheng-Wei Wu, and Philip S Yu. Efficient algorithms for mining high utility itemsets from transactional databases. *IEEE TKDE*, 25(8):1772–1786, 2013.

3. Ying Liu, Wei-keng Liao, and Alok Choudhary. A two-phase algorithm for fast discovery of high utility itemsets. In *Advances in Knowledge Discovery and Data Mining*, pages 689–695. Springer, 2005.

4. Bai-En Shie, Vincent S Tseng, and Philip S Yu. Online mining of temporal maximal utility itemsets from data streams. In *ACM Symposium on Applied Computing*, pages 1622–1626. ACM, 2010.

5. Hua-Fu Li, Hsin-Yun Huang, Yi-Cheng Chen, Yu-Jiun Liu, and Suh-Yin Lee. Fast and memory efficient mining of high utility itemsets in data streams. In *IEEE ICDM*, pages 881–886, 2008.

6. Raymond Chan, Qiang Yang, and Yi-Dong Shen. Mining high utility itemsets. In *IEEE ICDM*, pages 19–26, 2003.

7. Bai-En Shie, Hui-Fang Hsiao, Vincent S Tseng, and S Yu Philip. Mining high utility mobile sequential patterns in mobile commerce environments. In *DASFAA*, pages 224–238, 2011.

8. Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *20th VLDB*, volume 1215, pages 487–499, 1994.

9. Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *ACM SIGMOD Record*, volume 29, pages 1–12, 2000.

10. Carson Kai-Sang Leung, Quamrul I Khan, Zhan Li, and Tariqul Hoque. Cantree: A canonical-order tree for incremental frequent-pattern mining. *Knowledge and Information Systems*, 11(3):287–311, 2007.

11. Chowdhury Farhan Ahmed, Syed Khairuzzaman Tanbeer, Byeong-Soo Jeong, and Young-Koo Lee. Efficient tree structures for high utility pattern mining in incremental databases. *IEEE TKDE*, 21(12):1708–1721, 2009.

12. Vincent S Tseng, Cheng-Wei Wu, Bai-En Shie, and Philip S Yu. Up-growth: An efficient algorithm for high utility itemset mining. In *16th ACM SIGKDD*, pages 253–262, 2010.

13. Cheng Wei Wu, Bai-En Shie, Vincent S Tseng, and Philip S Yu. Mining top-k high utility itemsets. In *18th ACM SIGKDD*, pages 78–86. ACM, 2012.

14. Junfu Yin, Zhigang Zheng, Longbing Cao, Yin Song, and Wei Wei. Efficiently mining top-k high utility sequential patterns. In *IEEE ICDM*, pages 1259–1264. IEEE, 2013.

15. Tianjun Lu, Yang Liu, and Le Wang. An algorithm of top-k high utility itemsets mining over data stream. *Journal of Software*, 9(9):2342–2347, 2014.

16. B Goethals and MJ Zaki. the fimi repository, 2012.