Efficient Skyline Itemsets Mining

Vikram Goyal
IIIT Delhi, India
vikram@iiitd.ac.in

Ashish Sureka
Software Analytics Research Lab (SARL), India
ashish@iiitd.ac.in

Dhaval Patel
IIT Roorkee, India
patelfec@iitr.ac.in

ABSTRACT

Utility Mining (UM) in context of Market Basket Analysis consists of mining itemsets from a transaction database guided by optimizing utility. For example, UM consists of extracting all itemsets in a transaction database having utility above a user-defined minimum threshold or mining Top-K high utility itemset. Similarly, Frequent Itemset Mining (FIM) finds frequent patterns using a frequency threshold. However, none of these pattern mining methods determine patterns that are interesting in both the aspects of utility and frequency. In addition these methods require a user to specify respective thresholds. In this paper, we present a novel framework for mining a new pattern called as Utility-Frequency Skyline Pattern. We formalize our problem as a pattern search problem and propose an efficient technique on recently proposed popular data structure called as UP Tree (Utility-Pattern Tree). The proposed algorithm consists of two phases called as Filter and Refine. In the Filter phase, a set of candidate itemsets are mined, which are then verified finally in the Refine phase. We study the effectiveness of our proposed algorithm along with two heuristics and conclude that our proposed method is efficient.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Data mining

Keywords

Association Rule Mining, FP-Tree, Multi-Objective Optimization, Utility-Based Data Mining, Utility-Frequency Skyline Itemsets

1. INTRODUCTION

There has been done a lot of work done on both frequent itemset mining [1, 5] and high utility itemset mining [2, 4, 9, 7, 11, 6]. Extensive studies for both of these patterns show their interestingness with respect to many applications. For example, in the scenario of market basket data analysis, frequent itemsets mining algorithms discover itemsets that occur frequently in a database of transactions, whereas high utility itemsets mining algorithms discover itemsets with utility higher than a given threshold value in a transaction database. However, both of these patterns do not satisfy the requirement of a user who is interested in itemsets that are either frequently sold or result into high profit sales or both. For satisfying such user requirements, we define a new type of pattern called *Skyline* Frequent-Utility Itemsets.

Frequent itemsets mining has been an active research topic for many years. However, there may be many itemsets which are not frequent but have high utility. To fix this problem, a new type of itemsets pattern called high utility pattern was introduced in [3]. High utility pattern determines the itemsets with utility no less than a user defined threshold. Whereas, the utility of an item is defined using its interestedness (profitability or importance) value. The utility of items in a transaction is derived by multiplying two parameters: (1) the external utility of an item which refers to the item's importance, and (2) quantity of items in the transaction. An itemset is called high utility itemset if its aggregated utility over the set of transaction is above a given or pre-defined

Table 1: An Example Database

TID	Transactions	TU
1	(A,1) (C,1) (D,1) (H,1)	9
2	(A,2) (C,6) (E,2) (G,5)	27
3	(A,1) (B,2) (C,1) (D,6) (E,1) (F,5)	30
4	(B,4) (C,3) (D,3) (E,1)	20
5	(B,2) (C,2) (E,1) (G,2)	11
6	(A,1) (C,1) (D,1) (I,1)	33

Table 2: Item Utility Table

item	Ext. Utility
A	5
B	2
C	1
D	2
E	3
F	1
G	1
H	1
I	25

C3S2E 2015, July 13 - 15, 2015, Yokohama, Japan
© 2015 ACM. ISBN 978-1-4503-3419-8/15/07…$15.00
DOI: http://dx.doi.org/10.1145/2790798.2790816

threshold. Similar to frequent itemsets mining, high utility mining is also applicable to a wide range of applications such as marketing in stores, website click stream analysis. However, if a user is interested in both frequent and high utility itemsets then both these patterns alone fail to satisfy the requirement. In addition, all these techniques require a user to specify a threshold which is itself a difficult task.

In view of this, we introduce a new pattern called skyline frequent-utility (SFU) itemsets. An itemset X will be in SFU set if there is no other itemset in the database that has both higher utility as well as higher frequency than X. Consider a database of transactions consisting of items A, B, ..., F in Table 1 and items utilities in Table 2. Itemsets {C}, {A,C}, {A,C,D}, {B,C,D,E} will be SFU itemsets in our example, as there does not exist any other itemset X in the example database that dominate any of these itemsets in frequency and utility values.

However, mining SFU itemsets from a transaction database is a difficult task since it has to consider two dimensions together, i.e. frequency and utility, while classifying an itemset as a SFU. The difficulty arises due to existence of cases where an itemset is frequent but does not have high utility, and the cases where an itemset is of high utility but does not have high frequency. Existing work shows that frequent patterns have downward closure property which is violated in the case of high utility pattern, which makes searching of SFU pattern further difficult. A naive approach for discovering SFU pattern would be to first compute frequency and utility of each possible itemset and then find out the SFU itemsets, which is very computationally expensive.

To the best of our knowledge there has been no work done for mining this kind of pattern. However, application of existing methods for high frequent itemsets mining or high utility itemsets mining is not straightforward. We, therefore in this paper study SFU mining and present a framework to discover SFU itemsets. To address the efficiency issue, we give a set of pruning rules using utility estimates and propose a novel algorithm in the paper. The major contributions of this work are summarized as follows:

1. A novel pattern, called SFU pattern is defined that will satisfy the requirement of users who are interested in both frequent as well high utility patterns. A framework for mining SFU pattern is proposed.

2. Problem of mining SFU patterns is formalized as a pattern search problem and an efficient algorithm called SKYMINE is proposed.

3. Synthetic as well as real datasets are used in experiments to demonstrate the performance of SKYMINE algorithm. Two heuristics effective for frequent and high-utility itemset mining scenarios are also studied. The experimental results show that our proposed algorithm efficiently determines the correct and complete SFU.

2. RELATED WORK

Frequent itemset mining is an area that has attracted several researchers' attention and many papers on the topic has been published over the last decade. Initial algorithms for frequent itemset mining such as Apriori were based on the downward closure property (also called as the Apriori property) [1]. Han et al. propose a technique for mining frequent pattern without candidate generation. They propose a data structure called as FP-Tree (Frequent Pattern Tree) which is an extended prefix tree structure for storing information required by the depth-first search travel algorithm [5]. Other variants of itemset mining problem that have been proposed in the literature are High utility itemset mining, top-k frequent itemset mining and high utility-frequent itemset mining.

There are multiple algorithms proposed for high-utility itemset mining including Two-Phase [12], IHUP [2], UP-Growth [9] and UP-Growth+ [10]. IHUP and Two-Phase algorithms run in two phases where first phase generates candidate itemsets which are verified in the second phase. However due to large set of candidates generation in the first phase, both of these algorithms do not perform efficiently. On the other hand UP-Growth and UP-growth+ algorithms use an efficient data structure called UP-Growth tree (explained later in the section) to generate candidate itemsets whose size is comparatively much smaller than Two-Phase and IHUP kind of algorithms and hence achieve better performance. These algorithms use some heuristics namely DGU, DGN, DLU and DLN that prune out useless nodes and decrease utilities at nodes so as to result into less number of candidate itemset generation. We also use UP-tree for our algorithm and find skyline itemsets. However, none of the above algorithm can be directly used to determine SFU patterns.

For the case of top-k frequent pattern mining algorithms, several algorithms have been proposed. Most of the algorithms in this category follow some well defined search strategy to find top-k patterns with out missing any result pattern. In general they maintain a result set that is improved in each step of the algorithm until there is a guarantee of not finding any pattern with better frequency. The components of such algorithms are their search data structure and search strategies. We also model our solution in the form of a pattern search over a search data structure called UP-tree and find skyline candidates.

The work for high utility-frequent itemset mining finds patterns from the database that have their support frequency of transaction with threshold utility higher than the user defined frequency parameter. The authors in [12, 8] propose a concept of quasi-utility-frequency which is upward closed with respect to the lattice of all itemsets. They give a top-down two-phase algorithm where quasi-utility-frequency itemset candidates are generated in the first phase and then candidates are verified finally in second phase to generate utility-frequent itemset. However, this work also motivates the use of both frequency and utility dimension for itemset mining and restricts the transactions to be counted in for support value on the basis of utility value of an itemset X, but this can not be applied to determine skyline itemsets introduced by us. We present a new type of pattern that maximizes both the dimensions, namely utility and frequency, to generate non-dominating itemsets.

3. BACKGROUND

We give some definitions and define the problem of utility-frequency itemset mining formally in this section.

3.1 UP-Tree

A UP-Tree (Utility Pattern Tree) is a compact tree structure consisting of a header table. The header table contains

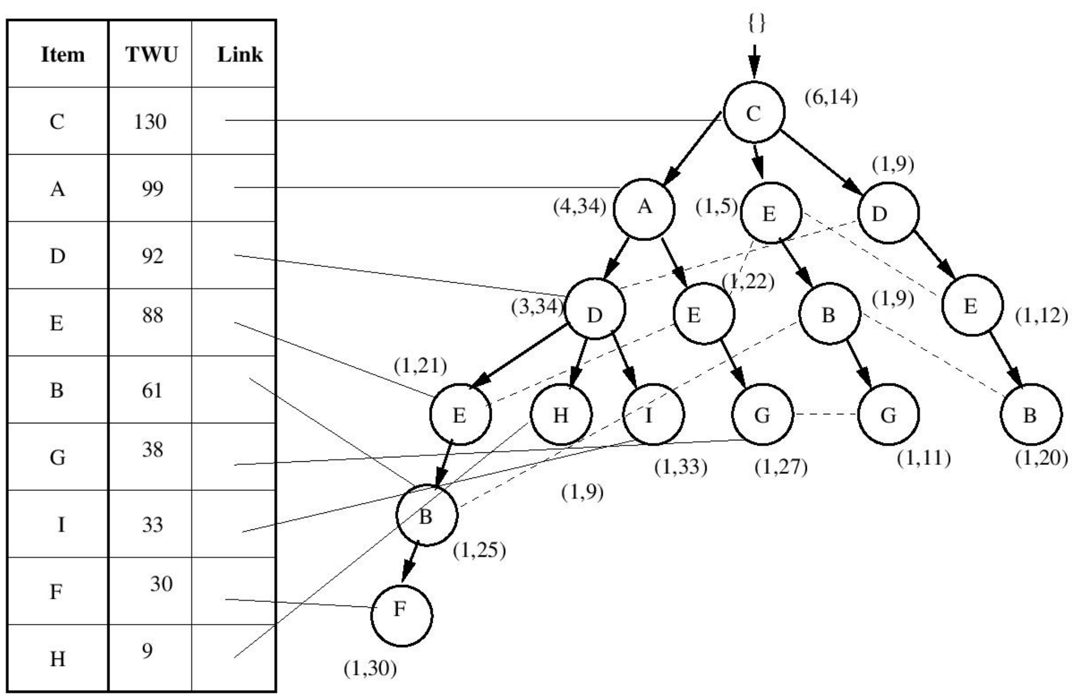

Figure 1: UP-Tree for the example database in Table 1

a list of items in the tree facilitating traversal of the UP-Tree nodes [9, 10]. The header table, for each item contains the item name, the estimated utility value and link to a node in the UP-Tree containing that item. The linked node from the header table is basically a start node to the linked list containing same item nodes and is used to traverse same item nodes in the UP-Tree. Each node n in the UP-Tree, except the root node (a special node that does not contain any item), has information about item name *n.name*, support count *n.count*, node utility *n.nu*, parent node *n.parent*, same item node link *n.hlink*. A node utility *n.nu* gives the estimate of the utility of an itemset comprising of items on the path from the node to the root node. Similarly, the value of support count at a node n, i.e., *n.count*, defines the support count of an itemset comprising of items on the path from that node to the root node [9]. The UP-Tree for the example database in Table 1 is shown in Figure 1.

3.2 Preliminary

Given a finite set of items $I = \{i_1, i_2, ..., i_m\}$, an itemset X is a set of k distinct items $\{i_1, i_2, ..., i_k\}$, where $i_j \in I, 1 \leq j \leq k$, and k is the length X. Each transaction T_d in the database of transactions $D = \{T_1, T_2, ..., T_n\}$, is an *itemset* and has associated with it a unique transaction identifier d. Each item i_p in a transaction T_d is also associated with a quantity $q(i_p, T_d)$, i.e., sold (purchased) quantity of i_p in T_d. Each item $i_p \in I$ has a unit profit value $pr(i_p)$ which a retailer is assumed to earn by selling a unit of item i_p.

Definition 1. *Item transaction utility of an item i_p in a transaction T_d, denoted by $u(i_p, T_d)$ is defined as the total profit earned through item i_p in transaction T_d, i.e., $u(i_p, T_d) = pr(i_p) \times q(i_p, T_d)$.*

For our example, $u(A, T_1) = 5 \times 1 = 5$.

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

For example, $u(\{A, C\}, T_1) = u(A, T_1) + u(C, T_1) = 5 + 1 = 6$.

Definition 3. *Utility of an itemset X in D is denoted as $u(X)$ and defined as $\sum_{T_d \in D}\{u(X, T_d) | X \subseteq T_d\}$.*

For example, $u(\{A, C\}) = u(\{A, C\}, T_1) + u(\{A, C\}, T_2) + u(\{A, C\}, T_3) + u(\{A, C\}, T_6) = 6 + 16 + 6 + 6 = 34$.

Definition 4. *Support count of an itemset X in D is denoted as $f(X)$ and defined as the number of transactions T_d in D containing X, i.e., $f(X) = | \{T_d \in D | X \subseteq T_d\} |$.*

For example, $f(\{A, C\}) = 4$.

Definition 5. *Let X and Y be two itemsets. Itemset X is said to be dominating itemset Y in D, denoted by $X \succ_D Y$ or $(f(X), u(X)) \succ (f(Y), u(Y))$, iff $f(X) \geq f(Y)$ and $u(X) > u(Y)$, or, $f(X) > f(Y)$ and $u(X) \geq u(Y)$.*

For our example database D, $\{A, C\} \succ_D \{A\}$ because $u(\{A, C\}) > u(\{A\})$ and $f(\{A, C\}) = f(\{A\})$.

Problem Statement. Given a transaction database D, mining utility-frequency skyline itemsets PS from a transaction database D is equivalent to discovering all itemsets X from D, which are not dominated by any itemset Y in D, i.e., $\forall X \in PS \nexists Y$ s.t. $Y \succ_D X$.

For our example, itemsets $\{C\}$, $\{A, C\}$, $\{A, C, D\}$ and $\{B, C, D, E\}$ are utility-frequency skyline itemsets.

Definition 6. *Given a database D and an index parameter j, the function $umin(j)$ returns the maximum utility value from utilities of itemsets having their support count greater than j, i.e. $umin(j) = max_X\{u(X) | f(X) > j\}$.*

Definition 7. *An itemset X with support count $f(X) = j$ is a candidate frequent-utility itemset if its utility is greater than $umin(j)$.*

For the example, $umin(4) = 14$, because $\{C\}$ is the only itemset having its frequency greater than 4. The utility value of $\{C\}$ is 14. The itemset $\{A, C\}$ is a candidate frequent-utility itemset as its support count is 4 and the utility value is 34.

The goal of utility-frequent itemset mining is equivalent to discovery of $umin(j)$ and candidate itemsets with highest utility value for each j, $1 \leq j \leq |D|$.

After giving the problem definition, we now introduce properties as introduced in earlier works [9].

Definition 8. *Transaction utility of a transaction T_d is denoted as $TU(T_d)$ and defined as $u(T_d, T_d)$.*

For our example, transaction utilities of each transaction is given in the column labeled TU of the Table 1.

Definition 9. *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 $\sum_{X \subseteq T_d \wedge T_d \in D} TU(T_d)$.*

The transaction weighted utility of $\{A, C\}$ is 99.

Definition 10. *Given a partially computed set of frequency-utility skyline itemsets CPS, an itemset X is called a potential frequency-utility skyline itemset (PFU) if $\nexists\, Y \in CPS$, s.t. $(f(Y), u(Y)) \succ_D (f(X), TWU(X))$.*

The transaction-weighted downward closure is denoted as *(TWDC)* and defined as follows: for any itemset X which is not a PFU, any superset of X is not a frequent-utility skyline itemset. In case X is not marked PFU, then every superset itemset of X will be dominated by some element in CPS.

4. MINING SKYLINE ITEMSETS

In this section, we first present a basic algorithm named *SKYMINE* for finding skyline frequent-utility (SFU) itemsets. The algorithm uses UP-tree for its working. We, then, present a couple of heuristics to improve the performance of SKYMINE algorithm.

4.1 SKYMINE Approach

There is currently no method proposed in the literature for mining SFU itemsets. The SKYMINE approach is basically consisted of two phases: i) generation of potential skyline itemsets (PFU) using UP-tree and, ii) identification of SFUs from PFUs.

Similar to other top-k frequent patterns mining methods [11], we follow a general process for mining PFU itemsets in the first phase of SKYMINE approach. Initially, a list L is created to store PFU itemsets ordered by their support counts. The utility threshold is set to 0 at each index j in the list, i.e., $umin(j) = 0$, $0 < j < |D|$. PFU patterns are searched from the database using a well defined search strategy. When a PFU pattern of a specific support count, say i, is found in the search process, it is added to the list L at index i. We associate a lower-bound (lb) and a upper-bound (ub) utility values with a PFU itemset which will be explained later in the section. After adding PFU itemset at index i, the list L is maintained in the following way: 1) We check $umin(j)$ values at lower indexes than i and the $umin(j)$ at index j, $0 \leq j \leq i < |D|$, is updated if $umin(j)$ is smaller than the lb utility value of the added PFU itemset 2) The PFU patterns at index i or lower than i are removed if they dominated by the added PFU itemset. A PFU itemset is said to be dominated if there exists another PFU itemset in the list with higher support value and higher lb utility value. The algorithm keeps on searching for more PFU patterns until no pattern is found by the search strategy.

4.2 Construction of UP-Tree

The process of creating a UP-tree is similar to previous approaches [9] with only being different in terms of determining unpromising items when applying Discarding Global Node Utilities (DGN) rule. The tree can be created in two database scans. The transaction utility and TWU of each item is computed in the first scan. The items are the inserted into the header table in the non-increasing order of their TWUs. Second database scan is used to insert the transactions in the UP-tree. Each transaction is first filtered out by removing unpromising items and then reorganized using the items TWUs. The reorganized transaction (RTU) is then inserted into the UP tree. A reorganized transaction $T_d = \{i_1, i_2, ..., i_m\}$ is inserted into the UP-tree using the function Insert_Reorganized_Transaction (N, i_x) as given in [9]. The function is performed as follows: each item i_x of transaction from begining to end is matched with the item of node N of tree starting from the root. In case item i_x matches, node utility and support count is updated by the prefix itemset utility value and one, respectively. If no node matches with the item i_x, a new node is created with node utility value set equal to transaction-prefix utility value and support to one.

4.3 Generating PFUs from the UP-Tree

The algorithm uses a list of $umin(j)$ values for its working. A $umin(j)$ value, as defined earlier, denotes the largest utility value of any itemset of support count greater than j. These values are then revised as and when new PFUs are added to the list L according to the process defined earlier. The method of determining a PFU itemset is based on the following lemma.

Let $U =< umin(1), umin(2), ..., umin(|D|) >$ be a list of $umin$ values. For any itemset X with support count value k, if $u(X) < umin(k)$ then X is not a SFU, else X is a PFU .

This is due to the reason of itemset X being dominated by an itemset in the list L, according to the definition of dominance and computation of $umin()$ values.

Definition 11. *(Optimal umin-thresholds U^*) Let R be the complete set of SFUs in D and umin-threshold U^*=($umin^*(0)$, $umin^*(1)$,..., $umin^*(|D|)$) is the set of umin() values derived from R. U^* is called the optimal umin-threshold if there does not exist any itemset $X \in D$ such that $u(X, D) > umin^*(f(X))$.*

After initial assignment of umin-threshold to 0 ($umin(j) = 0, |D| \geq j \geq= 0$), the algorithms performs UP-Growth search process to generate PFUs. During phase-1 of the algorithm an exact utility of an itemset is not known to decide whether

Algorithm 1 SKYMINEX()

```
1: if (TWU(X) ≥ umin(f(X)) and ub(X) ≥ umin(f(X)) then
2:    Add X to L[f(X)]
3:    Remove all itemsets Y at L[f(X)] s.t. $ub(Y) < lb(X)$
4:    umin(f(X)) = $max_{Y \in L[f(X)]}\{lb(Y)\}$
5:    for j=f(X) downto 1 do
6:       if $umin(j) < lb(X)$ then
7:          umin(j)= lb(X)
8:          Remove itemsets $Y \in L[j]$, s.t. $ub(Y) \leq lb(X)$
9:       end if
10:   end for
11: end if
```

the itemset is a PFU and to update the $umin(j)$ value. Thus, for an itemset X we use its ub value to determine whether the itemset is a PFU and use its lb value to update $umin(j)$ value, respectively. The lb and ub values are defined as follows

Definition 12. *The lower bound* (lb) *utility of an itemset* $X = \{a_1, a_2, ..., a_m\}$ *is defined as*

$$lb(X) = max\left(\sum_{i=1}^{m}(u_D(a_i) - mau(a_i) * (f(a_i) - f(X)), \sum_{1}^{m} f(X) * miu(a_i)\right) \quad (1)$$

Definition 13. *The upper bound* (ub) *utility of an itemset* $X = \{a_1, a_2, ..., a_m\}$ *is defined as*

$$lb(X) = min\left(\sum_{i=1}^{m}(u_D(a_i) - miu(a_i) * (f(a_i) - f(X)), \sum_{1}^{m} f(X) * mau(a_i)\right) \quad (2)$$

$mau(a)$ and $miu(a)$ are the maximum item utility and minimum item utility of an item a in database D, respectively. The $miu(a)$ of an item a is the value $u(a,t), t \in D$ such that $\nexists\, t' \in D$ and $u(a,t') < u(a,t)$. Similarly, $mau(a)$ of an item a is defined as the value $u(a,t), t \in D$ such that $\nexists\, t' \in D$ and $u(a,t') > u(a,t)$.

Let $S =< X_1, X_2, ..., X_n >$ be a set of partially computed PFU itemsets and U_S is the list of umin threshold values derived using lb values of X_is. For any itemset Y, if $ub(Y) < U_S[f(Y)]$, Y is not a SFU.

The SKYMINE algorithm 1 uses the UP-Growth process to generate a candidate itemsets X. Each candidate itemset X after generation is checked whether its TWU value is higher than the current $umin(f(X))$ value. If $TWU(X) < umin(f(X))$, X and all its supersets are not SFU itemsets. Otherwise, $ub(X)$ is compared with $umin(f(X))$ value. X is added to the list L if $ub(X)$ is higher. After adding X in L, the umin thresholds are revised using the lb value of X. X will remain in the list L for consideration in the second phase of algorithm if there does not exist any itemset $Y \in D$ such that $lb(Y) > ub(X)$ and $f(Y) \geq f(X)$. The algorithm continues searching for more PFUs until no candidate is found by UP-Growth.

The approach can be described as follows: for any newly mined itemset X using pattern growth search strategy, if its ub and TWU are higher than $umin(f(X))$, then lb of X can be used to update the list L and umin-thresholds U as given in 1.

4.4 Identifying SFUs from PFUs

After generating PFUs list L in the 1st phase of the algorithm, SFUs are identified by computing the exact utilities of PFUs. To compute exact utilities, the database D is scanned once more. The list L is then processed from the end to the begining and a parameter called $maxU$ is used to keep the current largest utility value seen by the algorithm. An itemset with lower utility value than the $maxU$ is removed while scanning the list.

5. HEURISTICS

In this section, we introduce a couple of heuristics to further improve the performance of SKYMINE algorithm.

5.1 Pre-processing of Singleton and Pair Itemsets (PSP)

The SKYMINE approach starts with umin threshold values of 0, which results into creation of full UP-Tree. Starting with a higher umin threshold values may result into pruning of some of the nodes of UP-tree. Similar to previous work done for association rule mining [1], we propose a heuristic called pre-processing of singleton and pairs (PSP). During the first scan of the database, utility of each single and pair itemset is evaluated. These singleton and pairs are then inserted into the list L one by one and only dominating itemsets remain in the list at the end. The umin thresholds are also updated according to the process described earlier.

5.2 Raising $umin(.)$ using Node utilities (RU)

The next proposed strategy is called *RU* in which we add itemsets described by nodes in the UP-Tree. This heuristic is applied using the nodes of UP-Tree constructed from the database D (not a local UP-Tree). Earlier works on UP-Tree [9] show that a node represents an itemset X consisting of items on nodes at the path from the node to the root node as well as the RTU value of the node is an lb value, i.e., $RTU(X) \leq u(X)$. With these results, we can use the node utilities and node support count to increase the $umin(.)$ threshold values. Increase in node utility value may help in pruning non-candidate items early and hence increase the performance of the algorithm.

6. EXPERIMENTS

Datasets: We have conducted extensive experiments to evaluate the proposed methods. All the experiments were performed on a computer with Intel Core-i3 CPU 2.10G Hz and 4.00 GB RAM. The operating system is Ubuntu11.04. All the algorithms are implemented in Java on Eclipse 3.5.2 platform with JDK 1.6.0_24. For performing experiments, we used different types of real world datasets, descriptions of which are given in Table 3.

The dataset *mushroom* was obtained from the FIMI Repository. It is dense dataset with about 8000 transactions each of which has about 23 items. The dataset *Foodmart* was acquired from Microsoft foodmart 2000 database. This dataset is sparse in nature with around 4000 transactions of length

Table 4: Experiment Results

Approach	Parameters	Datasets		
		Mushroom	Retail-store	Foodmart
SKY_{base}	#Candidates	1250367	36911	1629
	#Total-Time	217486 sec	26236	1.3 sec.
SKY_{prun}	#Candidates	5213	135622215	1174141
	#Total-Time	117 sec	33036	1.4 sec.

Table 3: Characteristics of Datasets

Dataset	$\#T_x$	*Avg.length*	*#Items*	*Type*
Mushroom	8,124	23.0	119	Dense
Foodmart	227	17.88	1559	Sparse
Retail Store	88,000	10.3	16469	Sparse

4 on the average. The third dataset we used for our experiment is Retail that we obtained from FIMI repositories. It is also a sparse dataset with around 88000 transaction with average transaction size of 10. The detailed description of these datasets are given in Table 3. The quantities of items are generated randomly with uniform distribution in the range of 1-6 in the case of mushroom and Retail datasets. The utility value for items is generated using log-normal distribution. The Foodmart dataset already contains quantity and utility values. The proposed strategy SKY is evaluated using three different variants, SKY_{base}, SKY_{prun}. SKY_{base} is the the version of the algorithm that does not use any sort of pre-computation for its working. On the other hand, SKY_{prun} algorithm use pre-processing and utility-raising techniques.

The performance results of algorithms SKY_{base} and SKY_{prun} are given in Table 4. It can be seen that SKY_{base} algorithm performs much better than a naive approach that would require to generate support and utility value of every possible combination of items. The use of pattern growth paradigm along with use of UP-growth data structure and tight estimates really help to prune the search space. It can also be observed that SKY_{base} outperforms SKY_{prun} in the case of sparse datasets, i.e. Retail-store and Foodmart. It is due to the reason of generating a large number of candidate itemsets as pairs and tree-node itemsets from sparse datasets, as the number of items in sparse dataset and nodes in UP-tree are very large. On the other hand, for dense datasets like Mushroom where the number of different items is small in number, the SKY_prun approach becomes very effective.

7. CONCLUSION

We introduce a new pattern called frequent-utility skyline pattern. This pattern is interesting as it relieves the user in terms of specifying different threshold parameters and return patterns that are relevant both in terms of frequency or utility. We have formalized the problem as a pattern search problem and use an existing data structure called UP-tree for its working. Through our experimental study, we have studied the impact of a couple of heuristics effective in the context of frequent-itemset mining and high-utility itemset mining, and observe that they still work good for the case of dense datasets like mushroom but perform poorly for sparse datasets.

8. REFERENCES

[1] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *20th VLDB*, pages 487–499, 1994.

[2] 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.

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

[4] Alva Erwin, RajP. Gopalan, and N.R. Achuthan. Efficient mining of high utility itemsets from large datasets. In *Advances in Knowledge Discovery and Data Mining*, volume 5012, pages 554–561, 2008.

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

[6] 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.

[7] Ying Liu, Wei-keng Liao, and Alok Choudhary. A fast high utility itemsets mining algorithm. In *1st International Workshop on Utility-based Data Mining*, pages 90–99, 2005.

[8] Vid Podpecan, Nada Lavrac, and Igor Kononenko. A fast algorithm for mining utility-frequent itemsets. *CONSTRAINT-BASED MINING AND LEARNING*, page 9, 2007.

[9] 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.

[10] V.S. Tseng, Bai-En Shie, Cheng-Wei Wu, and P.S. Yu. Efficient algorithms for mining high utility itemsets from transactional databases. *IEEE TKDE*, 25(8):1772–1786, Aug 2013.

[11] Cheng Wei Wu, Bai-En Shie, Vincent S. Tseng, and Philip S. Yu. Mining top-k high utility itemsets. In *18th ACM SIGKDD*, KDD '12, pages 78–86, 2012.

[12] Jieh-Shan Yeh, Yu-Chiang Li, and Chin-Chen Chang. Two-phase algorithms for a novel utility-frequent mining model. In *International Conference on Emerging Technologies in Knowledge Discovery and Data Mining*, pages 433–444, 2007.