ISSN: 0975-766X
CODEN: IJPTFI
Available Online through ***Research Article***
www.ijptonline.com

AN EFFICIENT MECHANISM TO MINE HIGH UTILITY ITEMSETS

Sarah Prithvika P.C.[1], Priya R.[2], Ramani S.[3] and Angel Princes A.[4]
[1,2,3,4] Assistant Professor, Department of Computer Science and Engineering, Panimalar Engineering College, Chennai, Tamil Nadu, India.
Email: sarah.chandran@gmail.com

Received on: 02-03-2017 *Accepted on: 05-04-2017*

Abstract

Mining high utility itemsets is useful because it provides a lot of profit to businesses. It is useful to increase the revenue of businesses because it is generated not only taking into consideration the frequency of itemsets but also takes into account other factors such as quantity and value. High utility itemsets can be mined using a minimum threshold value, but determining the minimum threshold value is difficult. If the value that is chosen is too low then too many itemsets will be categorized as high utility ones, if the value is too high, then only few itemsets will be found. Finding the minimum threshold value can be both challenging and tedious and it is found using a trial and error process. This paper discusses the methods to mine high utility datasets in an efficient manner without using the minimum threshold value.

Keywords: Utility mining, High-utility itemsets, Frequent itemset mining.

Introduction

Pattern mining is an important part of data mining. Pattern mining is used to uncover useful information from a large set of data. Frequent itemset mining [2], [3], [8], [4], [5], [9], [7], [6], [10] is used to discover frequently occurring itemsets. Frequent itemset mining will find frequently occurring data but it may be of low value, so we may miss some of the valuable data that have high value and low occurrence. So, utility mining is used to rectify this problem. Utility mining associates each item with a utility such as unit profit and the number of times it occurs in each transaction. An itemset is called high utility itemset (HUI) if its utility is not lesser than a user-specified minimum utility threshold. HUI mining is important to many applications such as retail marketing, mobile computing and Web services. It is difficult to mine HUIs in databases because the downward closure property does not hold [2], [3]. The downward closure property says that if there is a subset with infrequent itemsets, then its superset should not be considered or even generated. Therefore pruning the HUIs search space becomes difficult since a superset of a low utility itemset can be high utility. To overcome this problem the transaction weighted utilization (TWU) model [12] was used. An itemset can be termed as a high transaction-weighted utilization itemset (HTWUI) if its TWU is not lesser than a minimum utility threshold. The TWU of an itemset represents an upper limit on its utility. Therefore, a HUI must be a HTWUI. All the HUIs must be part of the complete set of HTWUIs. A transaction-weighted utilization concept based algorithm usually consists of two phases. In the first phase, all the sets of HTWUIs are discovered. In the second phase, all the HUIs are found by calculating the exact utilities of HTWUIs by searching the database. It difficult for users to choose a correct minimum utility threshold. If a low value is set for the minimum threshold utility then a large number of HUIs are generated resulting in memory constraints and waste of resources. If a high value is set for the minimum threshold utility then no HUIs are generated. This becomes a trial and error process where the algorithm is re-run until the desired results are obtained. So a better way as suggested in [1] is to allow the users to provide *k*, which is the number of desired itemsets, instead of specifying the minimum threshold value. Setting *k* is an easier task for the users because the user will have an idea of the number of itemsets that they want. Setting the minimum threshold value is difficult for the user as it depends on the characteristics of the database and the user usually does not know about it. Finding the top-*k* values is useful in many scenarios. If a user wants to know the top-*k* products that produce the highest profit, then by using this method they can find it. Developing algorithms to find the top-*k* HUIs is difficult and has some challenges.

The first challenge is that the utility of itemsets is neither monotone nor antimonotone [2], [3]. So, the utility of an itemset may be higher, lower or equal to that of its supersets and subsets. So, the same techniques developed in top-*k* frequent pattern mining that depend on the anti-monotonicity property cannot be used to prune the search space in top-*k* HUI mining. The second challenge is the problem of finding the exact utilities of itemsets in the first phase as it is unknown in the first phase. When a HTWUI is generated in phase I, we cannot guarantee that its utility is higher than other HTWUI and that it is a top-*k* HUI before the second phase. To ensure that all the top-*k* HUIs can be found in the set of high transaction-weighted utilization itemsets, a naive approach is to run the algorithm with minimum utility threshold set to 0. But this will make the search space to be large. The third challenge is that the minimum utility threshold is not given in advance in top-*k* HUI mining. So, the minimum utility is set to 0 and it is gradually increased to prune the search space. This threshold is called as the border minimum utility threshold min_util_{Border} [1]. It is different from the external parameter *min_util* that is given by users in advance. So, too many low utility itemsets would be produced in the intermediate level of the mining process if the threshold is not raised effectively and efficiently and thereby degrade its performance in terms of execution time and memory usage. Thus the challenge is to design effective strategies that can raise the *min_util* threshold as high and as quickly as possible, and further reduce as much as possible the number of candidates and intermediate low utility itemsets produced in the mining process.

The final challenge is how to raise the min_util_{Border} threshold effectively without missing any top-*k* HUIs. A good algorithm is one that can effectively raise the threshold in the mining process. Suppose an incorrect method for raising the threshold is used, it may result in some top-*k* HUIs being pruned. Thus, how to raise the threshold efficiently and effectively without missing any top-*k* HUI is a crucial challenge for this work.

Related Work

A. *High Utility Itemset Mining*

Many algorithms have been developed in the recent years to mine HUIs. Two phase algorithms generate potential candidates in the first phase and in the second phase they calculate the exact utility of each candidate found in phase I go find the HUIs. One phase algorithms find the HUIs in a single phase itself. d^2HUP [14] and HUI-Miner [13] are one-phase algorithms. A horizontal database is converted into a tree-based structure called CAUL [14] in d^2HUP. It discovers HUIs directly in databases. HUI-Miner considers a database of vertical format and transforms it into utility-lists [13]. The utility of generated itemsets is directly calculated in main memory without scanning the original database.

Let Table 1 be an example database containing five transactions. Each row in Table 1 indicates a transaction and each letter indicates an item and has an internal utility i.e. the quantity. The external utility (i.e. unit profit) of each item is shown in Table 2. If the *abs_min_util* = 30, then the HUIs in Table 1 is {{BD}:30, {BCD}:34, {BCE}:31, {BDE}:36, {BCDE}:40, the number beside each itemset is its absolute utility.

Table 1: An Example Database.

TID	Transaction	Transaction Utility (TU)
T_1	(A,1)(C,1)(D,1)	7
T_2	(A,2)(C,6)(E,2)(G,5)	25
T_3	(A,1)(B,2)(C,1)(D,6)(E,1)(F,4)	28
T_4	(B,4)(C,3)(D,3)(E,1)	20
T_5	(B,2)(C,2)(E,1)(G,1)	10

Table 2: Profit Table

Item	A	B	C	D	E	F	G
Unit Profit	60	58	90	55	83	28	35

The anti-monotone property cannot be directly used because the utility of an itemset may be higher, lower or equal to that of its supersets and subsets. Liu et al. introduced the concept of transaction-weighted downward closure property [12].

B. *Top-k Pattern Mining*

There are many kinds of top-*k* patterns and they differ in the type of patterns discovered, the search strategies employed and data structures. The search strategy and data structure chosen will affect the execution time and memory utilization efficiencies. Some of the algorithms are top-*k* frequent itemsets [8], [9], [7], top-*k* frequent closed itemsets [8], [6], top-*k* closed sequential patterns[15], top-*k* association rules [17], top-*k* sequential rules [16], top-*k* correlation patterns [18], [19], [20] and top-*k* cosine similarity interesting pairs[21].

C. *Top-k High Utility Pattern Mining*

Chan et al. [23] has worked in top-*k* high utility pattern mining but it did not take into consideration the quantitative values of the items. In [24], top-*k* HUI mining has been done by considering both quantities and profits of items. An algorithm T-HUDS has been proposed by Zihayat and An [25] for mining top-*k* HUIs over data streams. Yin et al. [26] have come up with a new way for mining top-*k* high utility sequential patterns. Ryang and Yun [24] proposed the extended REPT algorithm [22] with four strategies PUD, RIU, RSD and SEP. In REPT, along with *k* another parameter called N is needed, but it is very difficult to set this parameter.

The TKU Algorithm

A. TKU$_{Base}$ Algorithm

1) *UP-Tree Structure*

The TKU$_{\text{Base}}$ is extended from the UPGrowth [11]. TKU$_{\text{Base}}$ uses the UP-Tree structure of UP-Growth to keep the information of transactions and top-k HUIs. TKU$_{\text{Base}}$ is performed in three steps: (1) UP-Tree construction, (2) Generation of potential top-*k* HUIs (PKHUIs) from the UP-Tree, and (3) Identification of top-kHUIs from the set of PKHUIs. Every node N of a UP-Tree has five entries: Item name is denoted as N.name; Support Count is denoted as N. Count; Node utility is denoted as N.nu; N.parent indicates the parent node of N; Node link is denoted as N.hlink and it may point to a node having the same item name as that of N.Name. The Header table facilitates the traversal of the UP-Tree. The name of the item, the estimated value of the utility and a link are present in the header table entry. The link is made to point to the first node in the UP-Tree that has the same item name as the entry. The links in the header table and node links in the UP-Tree can be used for efficient traversal.

2) *Construction of UP-Tree*

A UP-Tree is constructed by scanning the database two times. In the first scan, the transaction utility of each transaction and TWU of each item are computed. For example in Table 3, we have the TWUs listed.

Table 3: Items and their TWUs.

Item	A	B	C	D	E	F	G
Unit Profit	60	58	90	55	83	28	35

Items are added into the header table in the descending order of their TWUs. During the second scan of the database, transactions are reordered and inserted into the UP-Tree. After reorganization a transaction is called are reorganized transaction (T_r’) and its transaction utility is called reorganized transaction utility (*RTU*). When a reorganized transaction T_r’ = $\{I_{1,}, I_{2,},...,I_M\}$ is retrieved $(I_j \in I^*, 1 \leq j \leq M)$, TKU$_{\text{base}}$ calls the function *Insert_Reorganized_Transaction* (N,I_j) and applies the strategy *Discarding Global Node utilities (DGN)* [11] to insert T_r’. The function *Insert_Reorganized_Transaction* takes a node *N* in the UP-Tree and an item I_j in the reorganized transaction as inputs. The function is performed as follows:

1. If there is a child node *ChN* for *N* and if *ChN.item*=I_j, then *ChN.count* is incremented by 1. Otherwise a new child node *ChN* is created with *ChN.parent*=*N*, *ChN.item*=$I_{j\,,}$ *ChN.count*=1 and *ChN.nu*=0.
2. Increase *ChN.nu* by (*RTU*(T_r’)- $\sum_{i=(j+1)}^{M} EU(I_j, T_r')$), where I_j T_r’ and $1 \leq j \leq M$,
3. Call *Insert_Reorganized_Transaction*(*ChN*,I_{j+1}) if $j \leq M$

   Once the reorganized transactions are inserted then the construction of UP-Tree is completed.

3) *Generating PKHUIs from the UP-Tree*

A minimum utility threshold is used in the TKU$_{\text{Base}}$ algorithm. It is called as border minimum utility threshold (denoted as *min_util*$_{Border}$). It is initially set to 0 and increased when a necessary number of itemsets with higher utilities has been found during the generation of PKHUIs.

*An itemset X is called top-*k *HUI (top-kHUI) in transactional database D if there are less than* k *itemsets whose utilities are larger than* EU(X) *in* $f_{HUI}(D, 0)$. EU(X) *is the absolute utility which is the product of internal and external utilities. So, if there exists* k *itemsets whose utilities are higher than the utility of itemset* Y, Y *is not a top-*k *HUI.* The minimum utility of an item *miu(I) = min{EU(I,T_R)|* $T_r \in D$ and $r \in g(I)\}$. The minimum utility of an item *miu(I) = min{EU(I,T_R)|* $T_r \in D$ and $r \in g(I)\}$.The minimum utility of an itemset X= $(I_1,I_2,\ldots\ldots I_M\}$ is *MIU(X)=* $\sum_{i=1}^{M} miu(I_i) \times SC(X)$. The maximum utility of an item is defined as *mau(I) = max{EU(I,T_r)|* $T_r \in D$, $r \in g(I)$.

A itemset is called Potential top-*k* HUI if its estimated utility value and MAU are no less than the min_util_{Border} threshold. TK_{Base} integrates the UP-Growth search process to discover top-*k* HUIs. The pseudo code for the TKU_{Base} algorithm is described in Algorithm 1.

Algorithm 1. TKU_{Base}

```
Input :  1. A database D
         2. Number of HUIs (k) needed
Output: The full set of PKHUIs C
Steps:
1. Set $min\_util_{Border} \leftarrow 0$;
   $TopK-MIU\text{-}List \leftarrow ø$;
   $C \leftarrow ø$
2. UP-Tree constructed by scanning D twice;
3. //Generate PKHUIs by applying a UP-
   Growth searching procedure
4. For each of the PKHUIs generated with
   estimated utility ESTU(X)  i.e. the estimated
   utility value do
5. {If(ESTU(X) $\geq \min\_util_{Border}$ and MAU(X)
   $\geq \min\_util_{Border}$)
6. {Output X and min{ESTU(X),MAU(X);
   $C \leftarrow C \cup X$ ;
7. If(MIU(X) $\geq \min\_util_{Border}$)
8. {Increase $min\_util_{Border}$ using strategy MC
9. $\min\_util_{Border} \leftarrow$
   $MC(MIU(X), TopK - MIU - List)$;
   }}}
```

To efficiently update min_util_{Border}, a structure called *TopK-MIU-List* is used to discover the *k* highest MIU values of PKHUIs found till that point of time. Once *k* MIU values are discovered and the *k-th* MIU value in *TopK-MIU-List* is greater than min_util_{Border}, min_util_{Border} is increased to

$MIU_{k\text{-}th}$ in *TopK-MIU-List*. If there are more than *k* MIU values in *TopK-MIU-List* then min_util_{Border} is raised to $MIU_{k\text{-}th}$ in *TopK-MIU-List* and the MIU values that are lesser than $MIU_{k\text{-}th}$ in *TopK-MIU-List* are deleted. The search is continued for other PKHUIs till no other candidate is found by the UP-Growth algorithm.

Strategy 1 (MC) – It involves raising the threshold by MIUs of candidates. If *MIU(X), ESTU(X)* and *MAU(X)* are not lesser than the current min_util_{Border} , then *MIU(X)* is added to *TopK-MIU-List*. Min_util_{Border} can be safely raised to $MIU_{k\text{-}th}$ if the | *TopK-MIU-List* | $\geq$ *k* and $MIU_{k\text{-}th} > min\_util_{Border}$.

4) Identifying Top-kHUIs from PKHUIs

The TKU_{Base} finds the utility of the PKHUIs by looking at the database to identify the top-*k* HUIs. A candidate itemset X is considered if its estimated utility value reached after phase I is not lesser than min_util_{Border}.

B. The TKU algorithm

Four strategies are discussed to raise the min_util_{Border} during different stages of data mining.

5) Pre-Evaluation

Min_util_{Border} is set to 0 before the construction of the UP-Tree in memory and this degrades the performance. So this scheme involves raising the min_util_{Border} before the construction of the UP-Tree and therefore pruning is easier and memory performance increases. Min_util_{Border} is raised during the first scan of the database.

Strategy 2 (PE: Pre-Evaluation) – This strategy uses a structure called as Pre-Evaluation Matrix(PEM) to store lower bounds of the utilities of certain 2-itemsets. *PEM[x][y]* is an entry in *PEM* and it corresponds to the lower bound of *EU({x,y}); x,y* $\in I^*$. At first, each entry in PEM is set to 0. When a transaction $T_r = \{I_1, I_2, \ldots, I_m\}, (I_j \in I^*, 1 \leq i \leq M)$ is obtained from the database, the utility of $\{I_1\} \cup \{I_i\}$ $1 < i \leq M$ in T_r is added to the value of the entry of $PEM[I_1][I_i]$ in *PEM*. If the *k-th* highest value in PEM is greater than min_util_{Border} then $min\text{-}util_{Border}$ is raised to the *k-th* highest value in *PEM*. Consider the database shown in Table 1. When T_1= {(A,1),(C,1),(D,1)} is obtained, the corresponding entries *PEM*[A][C], *PEM*[A][D] are included with *EU({AC},T1)*=5 and *EU({AD},T1)*=6. The other remaining transactions in the database are completed in the same manner. After that, if min_util_{Border} is lower than the *k-th* highest value in *PEM*, min_util_{Border} is set to the *k-th* highest value in *PEM*. It is to be noted that in TKU_{Base} the strategy *DGU* proposed in [11] cannot be used as the min_util_{Border} is set to 0 before the UP-Tree is constructed. If the strategy *PE* is applied to increase the min_util_{Border} , then *DGU* can be further used to prune those items whose TWUs are lesser than min_util_{Border}, which will in turn reduce the size of the UP-Tree and the number of candidates generated in phase I.

6) *Increasing the threshold by Node Utilities*

Strategy 3 (NU: the threshold is raised by node utilities)- This strategy is applied during the UP-Tree construction, during the second scan of the database. If there are more than *k* nodes in the current UP-Tree and the *k-th* highest node utility value $NU_{k\text{-}th}$ is higher thatn the min_util_{Border}, min_util_{Border} can be raised to $NU_{k\text{-}th}$.. Further pruning can be done by pruning the items whose TWU values are lesser than min_util_{Border} in the UP-Tree.

7) *Increasing the threshold by MIU Values of Descendents*

Strategy 4 (MD: increasing the threshold by MIU values of descendents) It is applied before the generation of PKHUIs and after the construction of the UP-Tree. Let N_{α} denote the node of the UP-Tree such that α denotes the item stored in *N*. For each node N_{α} under the UP-Tree root, the algorithm traverses the sub-tree under node N_{α} one time to find the support count of the itemset $\{\alpha\ U\ \beta\}$ for every descendent N_{β} of N_{α}. The MIU value of $\{\alpha\ U\ \beta\}$ is found for each itemset { α *U* β }. If the *k-th* highest MIU value is greater than min_util_{Border} then min_util_{Border} can be safely increased to that value.

8) *Increasing the threshold during Phase II*

Strategy 5 (SE: increasing the threshold by sorting and calculation of exact utility of candidates). This is applied in the phase II of TKU. If *C* is the set of candidates generated in Phase I, then the candidates in C are sorted in descending order of their estimated utilities.

So, candidates with greater estimated utility values will be taken into consideration before those with lower estimated utility values. During phase II, if the utility of a HUI *X* is greater than min_util_{Border}, *X* and *EU(X)* are included in the min-heap structure called *TopK-HUI-List*. HUIs in *TopK-HUI-List* are ordered in descending order of utilities. $Min\text{-}util_{Border}$ is raised to the utility of the *k-th* HUI in the *TopK-HUI-List* and HUIs having a utility lesser than the min_util_{Border} are removed from the list.

If *min {ESTU(Y), MAU(Y)}* is less than min_util_{Border} then Y and the other candidates need not be considered because the upper bound on their utilities is lesser than min_util_{Border}. So, the list *TopK-HUI-List* captures all the top-*k* HUIs in the database. So, in this scenario itemsets with lower estimated utility values may not be checked if the min_util_{Border} has been raised earlier.

The TKO Algorithm

In this algorithm the top-*k* HUIs can be found in one phase itself. The search procedure of HUI-Miner and its utility list structure [13] are used. The utility of an itemset that is generated by TKO is calculated by its utility-list without scanning of the database.

A. Construction of Utility-List Structure

Utility-lists are generated using [13]. In the TKO_{Base} and TKO algorithms each item is linked to a utility-list. These are called initial utility lists and these are obtained after scanning the database twice. During the first scan, the TWU and utility lists of items are calculated. During the second scan, items in each transaction are sorted in order of TWU values and the utility-list of each item is generated. The utility-list of an item set has more than one row. Each row has information of X in a transaction T_r. It has three fields: Tid, iutil and rutil. Fields Tid and iutil contain the identifier T_r and the utility of X in T_r. The field rutil indicates the remaining utility of X in T_r.

For example consider the table 4. The remaining utility of {D} in T_1 is $RU(\{D\},T1) = EU(\{A\},T1) + EU(\{C\}, T1)$ = (4+1) = 5. The remaining utility of {D} in the database is $RU(\{D\})=RU(\{D\},T1) + RU(\{D\},T3) + RU(\{D\},T4)$ = (5+17+14) = 36. The sample utility lists for G and F are found in Fig.1 .

Table 4: Transactions to generate utility-lists.

TID	Transaction	Transaction Utility (TU)
T_1	(D,1)(A,1)(C,1)	7
T_2	(G,5)(A,2)(E,2)(C,6)	25
T_3	(F,5)(D,6)(B,2)(A,1)(E,1)(C,1)	29
T_4	(D,3)(B,4)(E,1)(C,3)	20
T_5	(G,2)(B,2)(E,1)(C,2)	11

Fig 1. Sample utility lists.

{G}		
2	5	20
5	2	9

{F}		
3	5	24

D. The TKO_{Base} Algorithm

The TKO_{Base} algorithm takes as input the parameter *k* and a database in horizontal format. Suppose a database has already been transformed into vertical format such as initial utility-lists, TKO_{Base} can be directly used for mining top-*k*-HUIs.

TKO_{Base} initially sets the min_util_{Border} threshold to 0 and initializes a min-heap structure known as *TopK-CI-List* for keeping a track of the current top-*k* HUIs during a search. The algorithm scans the database two times to build the initial utility-lists. Then TKO_{Base} explores the search space of top-*k* HUIs using a procedure that we call as *TopK-HUI-Search*. It combines *RUC* (Raising threshold by utility of candidates) and *HUI-Miner*[13]. TKO_{Base} updates the list of top-*k* HUIs in *TopK-CI-List* and raises the min_util_{Border} threshold by the information of *TopK-CI-List*. Thus the *TopK-CI-List* captures the *top-k* HUIs in the database. Algorithm. 2 shows the pseudo code of *TopK-HUI-Search* procedure.

Algorithm 2. TopK-HUI-Search

```
Input:  u(P) :utility-list for a prefix P
        Class[P]: a set of itemsets with respect to the
                  prefix P
        ULS[P]:a set of utility-lists with respect to the
                  prefix P
        $\delta$ : border minimum utility threshold
             $min\_util_{Border}$
        TopK-CI-List : a list for maintaining candidate
                  itemsets
Output: Use TopK-CI-List to find all the top-k HUIs
Steps:
        1. For each $X= (x_1,x_2,\ldots\ldots x_L\} \in Class[P]$ do
        2. { if($SUM(X.iutils) \geq \delta$)
        3. { //Increase $min\_util_{Border}$ by RUC
        4. $\delta \leftarrow RUC(X,TopK\text{-}CI\text{-}LIST)$;}
5. if$(SUM(X.iutils)+SUM(X.rutils) \geq \delta)$
6.{$Class[X] \leftarrow$ ø; $ULS[X] \leftarrow$ ø;
7. For each $Y=\{y_1,y_2,\ldots\ldots y_L\} \in Class[P] \mid$
    $y_L > x_L$ do
8. {$Z \leftarrow XUY$;
9. $ul(Z) \leftarrow Construct(ul(P),X,Y,ULS[P])$;
10. $Class[X] \leftarrow Class[X]\ U\ Z$;
11. $ULS[X] \leftarrow ULS[X]\ U$ ul(Z);}
12. $TopK\text{-}HUI\text{-}SEARCH(X,ULS[X], Class[X],$
    $\delta$,$TopK\text{-}CI\text{-}List)$;}}
```

Strategy 6 (RUC : Raising the threshold by the utilities of candidates) – This strategy can be used in any one-phase algorithm where itemsets are found with their utilities. It uses the *TopK-CI-List* structure to maintain Top-*k* HUIs, where itemsets are sorted in descending order of their utilities. Initially, *TopK-CI-List* is empty. When an itemset *X* is found by the search procedure and its utility is not lesser than min_util_{Border}, then *X* is added to *TopK-CI-List*. If there are more than *k* itemsets in *TopK-CI-List*, min_util_{Border} can be increased to the utility of the *k-th* itemset in *TopK-CI-List*. After that the itemsets that have a utility lesser than the increased min_util_{Border} are removed from the *TopK-CI-List*.

C. The TKO Algorithm

The TKO algorithm improvises the TKO_{Base} algorithm by including four strategies. *PE* and *DGU* were discussed earlier, below are the other two strategies.

Strategy 7: *RUZ (Reducing estimate utility values by using Z-elements).* For any candidate X generated by the *TKO* algorithm, it is not necessary to explore the search space of concatenations of *X* if *[NZEU(X) + RU(X)]* is lesser than min_util_{Border}.

Strategy 8: *EPB (Exploring the most promising branches first)*. This strategy aims to generate candidate itemsets with higher utility first. The concept is to extend the itemset having the largest estimated utility value first as it has greater chances of generating itemsets having a higher utility. So, it allows for increasing the min_util_{Border} more quickly for easy pruning of the search space.

Performance Evaluation

Experiments were performed using the TKU and TKO algorithms using both real and synthetic datasets with sparse, sense and large datasets [1].TKU was compared to UP-Growth[11] which is one of the current best 2 phase HUI mining algorithms. TKO was compared with REPT [22], TKU and TKO algorithms. It was found that the performance was close to the optimal case of the currently popular two-phase and one-phase mining algorithms. The algorithms also had a good scalability on large datasets.

Conclusion

The problem of mining top-*k* HUIs was discussed. Two algorithms TKU (mining Top-*k* utility itemsets in two phase) and TKO (mining top-*k* utility itemsets in one phase) were [1] discussed. TKU uses five strategies PE, NU, MD, MC and SE to raise the border minimum utility threshold and prune the search space. TKO uses the strategies RUC, RUZ and EPB. Both algorithms explore effective strategies to efficiently prune the search space.

References

1. Vincent S. Tseng, Cheng-Wei Wu, Philippe Fournier-Viger and Philip S. Yu, “Efficient Algorithms for Mining Top-K High Utility Itemsets”, IEEE Transactions on Knowledge and Data Engineering, Vol. 28, No. 1, January 2016.
2. R. Agrawal and R. Srikant, “Fast algorithms for mining association rules,” in Proc. Int. Conf. Very Large Data Bases, 1994, pp. 487–499.
3. J. Han, J. Pei, and Y. Yin, “Mining frequent patterns without candidate generation,” in Proc. ACM SIGMOD Int. Conf. Manag. Data, 2000, pp.1–12.
4. J. Han, J. Wang, Y. Lu, and P. Tzvetkov, “Mining top-k frequent closed patterns without minimum support,” in Proc. IEEE Int. Conf. Data Mining, 2002, pp. 211–218
5. J. Pisharath, Y. Liu, B. Ozisikyilmaz, R. Narayanan, W. K. Liao, A. Choudhary, and G. Memik, NU-MineBench version 2.0 dataset and technical report [Online]. Available: http://cucis.ece.northwestern.edu/projects/DMS/MineBench.html, 2005.
6. J. Wang and J. Han, “TFP: An efficient algorithm for mining top-k frequent closed itemsets,” IEEE Trans. Knowl. Data Eng., vol. 17, no. 5, pp. 652–663, May 2005.
7. T. Quang, S. Oyanagi, and K. Yamazaki, “ExMiner: An efficient algorithm for mining top-k frequent patterns,” in Proc. Int. Conf. Adv. Data Mining Appl., 2006, pp. 436 – 447.

8. K. Chuang, J. Huang, and M. Chen, “Mining top-k frequent patterns in the presence of the memory constraint,”VLDB J., vol. 17, pp. 1321–1344, 2008.
9. G. Pyun and U. Yun, “Mining top-k frequent patterns with combination reducing techniques,”Appl. Intell., vol. 41, no. 1, pp. 76–98, 2014.
10. C. Wu, Y. Lin, P. S. Yu, and V. S. Tseng, “Mining high utility episodes in complex event sequences,” in Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, 2013, pp. 536–544
11. V. S. Tseng, C. Wu, B. Shie, and P. S. Yu, “UP-Growth: An efficient algorithm for high utility itemset mining,” inProc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, 2010, pp. 253–262
12. Y. Liu, W. Liao, and A. Choudhary, “A fast high utility itemsets mining algorithm,” in Proc. Utility-Based Data Mining Workshop, 2005, pp. 90–99.
13. M. Liu and J. Qu, “Mining high utility itemsets without candidate generation,” in Proc. ACM Int. Conf. Inf. Knowl. Manag., 2012, pp. 55– 64.
14. J. Liu, K. Wang, and B. Fung, “Direct discovery of high utility itemsets without candidate generation,” in Proc. IEEE Int. Conf. Data Mining, 2012, pp. 984–989.
15. P. Tzvetkov, X. Yan, and J. Han, “TSP: Mining top-k closed sequential patterns,”Knowl. Inf. Syst., vol. 7, no. 4, pp. 438–457, 2005
16. P. Fournier-Viger and V. S. Tseng, “Mining top-k sequential rules,” in Proc. Int. Conf. Adv. Data Mining Appl., 2011, pp. 180–194.
17 . Fournier-Viger, C. Wu, and V. S. Tseng, “Mining top-k association rules,” in Proc. Int. Conf. Can. Conf. Adv. Artif. Intell., 2012, pp. 61–73.
18. H. Xiong, M. Brodie, and S. Ma, “TOP-COP: Mining TOP-K strongly correlated pairs in large databases,” in Proc. IEEE Int. Conf. Data Mining, 2006, pp. 1162–1166.
19. H. Xiong, P. Tan, and V. Kumar, “Mining strong affinity association patterns in data sets with skewed support distribution,” in Proc. IEEE Int. Conf. Data Mining, 2003, pp. 387–394.
20. H. Xiong, P. Tan, and V. Kumar, “Hyperclique pattern discovery,” Data Mining Knowl. Discovery, vol. 13, no. 2, pp. 219–242, 2006.
21. S. Zhu, J. Wu, H. Xiong, and G. Xia, “Scaling up top-k cosine similarity search,”Data Knowl. Eng., vol. 70, no. 1, pp. 60–83, 2011.

22. H. Ryang and U. Yun, "Top-k high utility pattern mining with effective threshold raising Strategies,"Knowl.-Based Syst., vol. 76, pp.109–126,2015
23. R. Chan, Q. Yang, and Y. Shen, "Mining high-utility itemsets," in Proc. IEEE Int. Conf. Data Mining, 2003, pp. 19–26.
24. C. Wu, B. Shie, V. S. Tseng, and P. S. Yu, "Mining top-k high utility itemsets," in Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, 2012, pp. 78–86.
25. M. Zihayat and A. An, "Mining top-k high utility itemsets over data streams," Inf. Sci., vol. 285, no. 20, pp. 138–161, 2014.
26. J. Yin, Z. Zheng, L. Cao, Y. Song, and W. Wei, "Mining top-k high utility sequential patterns," in Proc. IEEE Int. Conf. Data Mining, 2013, pp. 1259–1264.