

An Efficient Candidate Pruning Technique for High Utility Pattern Mining

Chowdhury Farhan Ahmed, Syed Khairuzzaman Tanbeer,
Byeong-Soo Jeong, and Young-Koo Lee

Department of Computer Engineering, Kyung Hee University
1 Seochun-dong, Kihung-gu, Youngin-si, Kyunggi-do, 446-701, Republic of Korea
{farhan,tanbeer,jeong,yklee}@khu.ac.kr

Abstract. High utility pattern mining extracts more useful and realistic knowledge from transaction databases compared to the traditional frequent pattern mining by considering the non-binary frequency values of items in transactions and different profit values for every item. However, the existing high utility pattern mining algorithms suffer from the level-wise candidate generation-and-test problem and need several database scans to mine the actual high utility patterns. In this paper, we propose a novel tree-based candidate pruning technique HUC-Prune (high utility candidates prune) to efficiently mine high utility patterns without level-wise candidate generation-and-test. It exploits a pattern growth mining approach and needs maximum three database scans in contrast to several database scans of the existing algorithms. Extensive experimental results show that our technique is very efficient for high utility pattern mining and it outperforms the existing algorithms.

Keywords: Data mining, knowledge discovery, frequent pattern mining, high utility pattern mining, association rules.

1 Introduction

Frequent pattern mining [1], [5], [8] plays an important role in data mining and knowledge discovery techniques such as association rule mining, classification, clustering, time-series mining, graph mining, web mining etc. The initial solution of frequent pattern mining, level-wise candidate set generation-and-test paradigm of *Apriori* [1] has revealed many drawbacks that it requires multiple database scans and generates lots of candidate patterns. FP-growth [5] solved this problem by introducing a prefix-tree (FP-tree) based algorithm without candidate set generation-and-test. However, frequent pattern mining has two principal limitations. First, it treats all the items have the same importance/weight/price and second, in one transaction each item appears in a binary (0/1) form, i.e. either present or absent. But in our real world scenarios, each item in the supermarket has different importance/price and one customer may buy multiple copies of an item. Therefore, frequent patterns only reflect the correlation between items, and it does not reflect the semantic significance of the items.

A high utility mining [2], [3], [4], [6], [7] model was defined to solve the above limitations of frequent pattern mining. It allows the user to conveniently measure the importance of an itemset by the utility value. By utility mining, several important decisions in business area like maximizing revenue, minimizing marketing or inventory costs can be taken and more important knowledge about itemsets/customers contributing to the majority of the profit can be discovered. Other application areas, such as biological gene database, web click streams, stock tickers, network traffic measurements, web-server logs, data feeds from sensor networks and telecom call records can have similar solutions.

The existing high utility pattern mining algorithms [2], [3], [4], [7] suffer from the level-wise candidate generation-and-test problem. Therefore, they generate a huge number of candidates and need several database scans to mine the actual high utility patterns. Moreover, their number of database scans is directly dependent on the maximum length of the candidate patterns. In this paper, we propose a novel tree-based candidate pruning technique HUC-Prune (high utility candidates prune) to efficiently mine high utility patterns without level-wise candidate set generation-and-test. It prunes a large number of unnecessary candidates during the mining process. It exploits a pattern growth mining approach and needs a maximum of three database scans in contrast to several database scans of the existing algorithms. Extensive experimental results show that our technique is very efficient for high utility pattern mining and it outperforms the existing algorithms.

The remainder of this paper is organized as follows. In Section 2, we describe the related work. In Section 3, we describe the high utility pattern mining problem. In Section 4, we describe our proposed pruning technique HUC-Prune to efficiently mine high utility patterns. In Section 5, our experimental results are presented and analyzed. Finally, in Section 6, conclusions are drawn.

2 Related Work

The theoretical model and definitions of high utility pattern mining were given in [2], named MEU (mining with expected utility). They cannot maintain the *downward closure* property of *Apriori*. They used a heuristic to determine whether an itemset should be considered as a candidate itemset. It usually overestimates, especially at the beginning stages, where the number of candidates approaches the number of all the combinations of items. This is impractical whenever the number of distinct items is large and utility threshold is low. Later, the same authors proposed two new algorithms UMining and UMining_H [3] to calculate the high utility patterns. In UMining they have used a pruning strategy based on utility upper bound property. UMining_H has been designed with another pruning strategy based on a heuristic method. But, some high utility itemsets may be erroneously pruned by their heuristic method. Moreover, these methods do not satisfy the *downward closure* property of *Apriori* and also suffer from the level-wise candidate generation-and-test methodology.

The Two-Phase [4] algorithm was developed based on the definitions of [2] to find high utility itemsets using the *downward closure* property. They have defined “transaction weighted utilization”, and proved it can maintain the *downward closure* property. For the first database scan, their algorithm finds all the 1-element transaction weighted utilization itemset and based on that generates the candidates for 2-element transaction weighted utilization itemsets. In the second database scan, it finds all the 2-element transaction weighted utilization itemset and based on that generates the candidates for 3-element transaction weighted utilization itemsets and so on. At the last scan, it finds out the actual high utility itemsets from high transaction weighted utilization itemsets. This algorithm suffers from the same problem of the level-wise candidate generation-and-test methodology. CTU-Mine [6] proposed an algorithm that is efficient over Two-Phase algorithm only in dense database when the minimum utility threshold is very low.

The isolated items discarding strategy (IIDS) [7] for discovering high utility itemsets was proposed to reduce some candidates in every pass of databases. They developed efficient high utility itemset mining algorithm FUM and DCG+ and showed that their work is better than all previous high utility pattern mining works. But still their algorithms suffers from the level-wise candidate generation-and-test problem and needs multiple database scans depending on the maximum length of the candidate patterns. Therefore, we propose a novel tree-based pruning technique to remove these problems of the existing works.

3 Problem Definition

We have adopted similar definitions presented in the previous works [2], [3], [4]. Let $I = \{i_1, i_2, \dots, i_m\}$ be a set of items and D be a transaction database $\{T_1, T_2, \dots, T_n\}$ where each transaction $T_i \in D$ is a subset of I . The local transaction utility value $l(i_p, T_q)$, represents the quantity of item i_p in transaction T_q . For example, in Fig. 1(a), $l(b, T_2) = 6$. The external utility $p(i_p)$ is the unit profit value of item i_p . For example, in Fig. 1(b), $p(c) = 3$. Utility $u(i_p, T_q)$, is the quantitative measure of utility for item i_p in transaction T_q , defined by

$$u(i_p, T_q) = l(i_p, T_q) \times p(i_p) \tag{1}$$

For example, $u(b, T_1) = 2 \times 6 = 12$ in Fig. 1. The utility of an itemset X in transaction T_q , $u(X, T_q)$ is defined by,

$$u(X, T_q) = \sum_{i_p \in X} u(i_p, T_q) \tag{2}$$

where $X = \{i_1, i_2, \dots, i_k\}$ is a k-itemset, $X \subseteq T_q$ and $1 \leq k \leq m$. For example, $u(bc, T_1) = 2 \times 6 + 8 \times 3 = 36$ in Fig. 1. The utility of an itemset X is defined by,

$$u(X) = \sum_{T_q \in D} \sum_{i_p \in X} u(i_p, T_q) \tag{3}$$

TID \ ITEM	a	b	c	d	e	Trans. Utility(\$)
T ₁	0	2	8	2	0	52
T ₂	4	6	0	0	0	44
T ₃	0	3	0	0	2	38
T ₄	2	2	7	0	0	37
T ₅	6	5	0	4	0	74
T ₆	4	4	0	5	3	102
T ₇	0	0	0	3	4	64
T ₈	0	0	0	2	0	16

(a) Transaction database

ITEM	PROFIT(\$ (per unit)
a	2
b	6
c	3
d	8
e	10

(b) Utility table

Fig. 1. Example of a transaction database and utility table

For example, $u(ab) = u(ab, T_2) + u(ab, T_4) + u(ab, T_5) + u(ab, T_6) = 44 + 16 + 42 + 32 = 134$ in Fig. 1. The transaction utility of transaction T_q denoted as $tu(T_q)$ describes the total profit of that transaction and it is defined by,

$$tu(T_q) = \sum_{i_p \in T_q} u(i_p, T_q) \tag{4}$$

For example, $tu(T_1) = u(b, T_1) + u(c, T_1) + u(d, T_1) = 12 + 24 + 16 = 52$ in Fig. 1. The minimum utility threshold δ , is given by the percentage of total transaction utility values of the database. In Fig. 1, the summation of all the transaction utility values is 427. If δ is 25% or we can also express it as 0.25, then the minimum utility value can be defined as

$$minutil = \delta \times \sum_{T_q \in D} tu(T_q) \tag{5}$$

Therefore, in this example $minutil = 0.25 \times 427 = 106.75$ in Fig. 1. An itemset X is a high utility itemset, if $u(X) \geq minutil$. Finding high utility itemsets means determining all itemsets X having criteria $u(X) \geq minutil$. The main challenge of facing high utility pattern mining areas is the itemset utility does not have the *downward closure* property. For example, if $minutil = 106.75$ in Fig. 1, then “e” is a low utility item as $u(e) = 90$. However, “de” is a high utility itemset as $u(de) = 134$. So, the *downward closure* property is not satisfied. We can maintain the *downward closure property* by transaction weighted utilization. Transaction weighted utilization of an itemset X , denoted by $twu(X)$, is the sum of the transaction utilities of all transactions containing X .

$$twu(X) = \sum_{X \subseteq T_q \in D} tu(T_q) \tag{6}$$

For example, $twu(bc) = tu(T_1) + tu(T_4) = 52 + 37 = 89$ in Fig. 1. Here, for $minutil = 106.75$ in Fig. 1 as $twu(bc) < minutil$, any super pattern of “bc” cannot be a high twu itemset (candidate itemset) and obviously cannot be a high utility itemset. X is a high transaction weighted utilization itemset (i.e. a candidate itemset) if $twu(X) \geq minutil$.

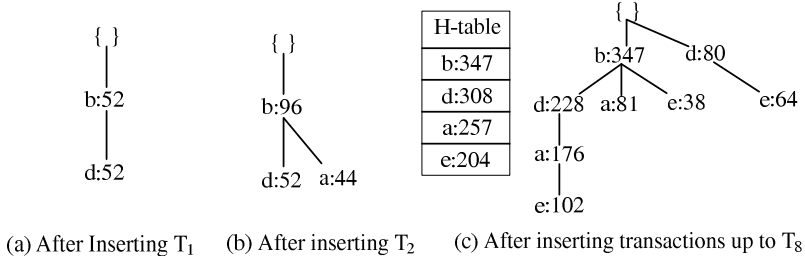


Fig. 2. Tree construction process

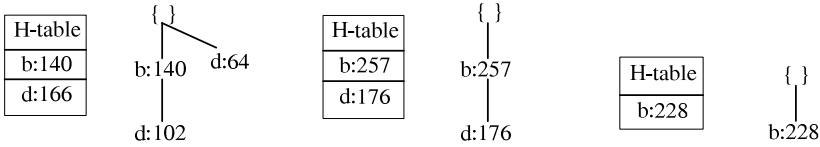
4 HUC-Prune: Our Proposed Technique

At first, we describe the construction process of our tree structure. Header table is maintained in our tree structure. Each entry in a header table and node of the tree explicitly maintain item-id and *twu* (transaction weighted utilization) value for each item. To facilitate the tree traversals adjacent links are also maintained (not shown in the figures for simplicity) in our tree structure. In the first database scan HUC-Prune captures the *twu* value of all the items in order to prune unnecessary candidates. We have explained in Section 3 that the *downward closure* property can be maintained by using the *twu* value. Therefore, by pruning the single-element items having *low-twu* value with respect to the given threshold, HUC-Prune achieves huge gain in tree-based candidate generation process.

Consider the database shown in Fig. 1 and *minutil* = 106.75. After the first database scan, the *twu* values are $\langle a:257, b:347, c:89, d:308, e:204 \rangle$. As a result, item “c” is a low *twu* item and according to the *downward closure* property its any superset cannot be high utility itemset. Therefore, we prune this item. Next, we sort the header table in descending order according to their *twu* values and the new order is $\langle b:347, d:308, a:257, e:204 \rangle$. In the second database scan, we take only high *twu* items from the transactions, sort them according to the header table order, and insert them into the tree. For the first transaction T_1 , which contains item “b”, “c” and “d”, we discard the low *twu* item “c” and arrange it according to the header table order. Both “b” and “d” get the *tu* value of T_1 (the *tu* value of T_1 is 52, shown in Fig. 1). Fig. 2(a) shows the tree after inserting T_1 . After that T_2 is inserted in the tree (shown in Fig. 2(b)). Before insertion the items of T_2 are arranged (at first “b” then “a”) in the header table order. Item “b” gets the prefix sharing with the existing node containing item “b”. Its *twu* value becomes $52+44=96$, and item “a” becomes its child with *twu* value of 44. Fig. 2(c) shows the final tree with the header table for the full database presented at Fig. 1. The following property is true for our tree structure.

Property 1. The total count of *twu* value of any node in the tree is greater than or equal to the sum of total counts of *twu* values of its children.

Now, we describe the mining process of our proposed HUC-Prune technique. As our tree-structure has the important property of FP-tree stated in property



(a) Conditional-tree for item “e” (b) Conditional-tree for item “a” (c) Conditional-tree for item “d”

Fig. 3. Mining process of HUC-Prune

1, pattern growth mining algorithm can be directly applicable to it by using the *twu* value. Consider the database of Fig. 1. The final tree is created for that database is shown in Fig. 2(c). If we take $\delta = 0.25$ in that database, then $minutil = 106.75$ according to equation 5. As like FP-growth, we have started from the bottom-most item.

At first the conditional tree of the bottom-most item “e” (shown in Fig. 3(a)) is created by taking all the branches prefixing the item “e” and deleting the nodes containing an item which cannot be a candidate pattern (high *twu* pattern) with the item “e”. Obviously, item “a” cannot be a candidate itemset with item “e” as it has low *twu* value with the item “e”. So, the conditional tree of item “e” does not contain the item “a”. However, candidate patterns (1) $\{b, e: 140\}$, (2) $\{d, e: 166\}$, (3) $\{e: 204\}$ are generated for the item “e”. In the similar fashion, conditional tree for item “a” is created in Fig. 3(b) and candidate patterns (4) $\{a, b: 257\}$, (5) $\{a, d: 176\}$, (6) $\{a, b, d: 176\}$, (7) $\{a: 257\}$ are generated. After that, conditional tree for item “d” is created in Fig. 3(c), and candidate patterns (8) $\{b, d: 228\}$, (9) $\{d: 308\}$ are generated. The last candidate pattern (10) $\{b: 347\}$ is generated for the topmost item “b”. Third database scan is required to find high utility itemsets from these 10 candidate high *twu* itemsets. The high utility 6 itemsets are $\{a, b: 134\}$, $\{a, b, d: 146\}$, $\{b: 132\}$, $\{b, d: 154\}$, $\{d: 128\}$ and $\{d, e: 134\}$.

5 Experimental Results

To evaluate the performance of our proposed technique, we have performed several experiments on IBM synthetic T10I4D100K dataset and real-life mushroom dataset from frequent itemset mining dataset repository (<http://fimi.cs.helsinki.fi/data/>). These datasets do not provide the profit values or quantity of items in transactions. As like the performance evaluation of the previous utility based pattern mining [4], [6], [7] we have generated random numbers for the profit values of each item and quantity of each item in each transaction, ranging from 0.01 to 10.0 and 1 to 10 respectively. Observed from real world databases that most of the items carry low profit, we generate the profit values using a lognormal distribution [4], [6], [7]. Our programs were written in Microsoft Visual C++ 6.0 and run with the Windows XP operating system on a Pentium dual core 2.13 GHz CPU with 1GB main memory.

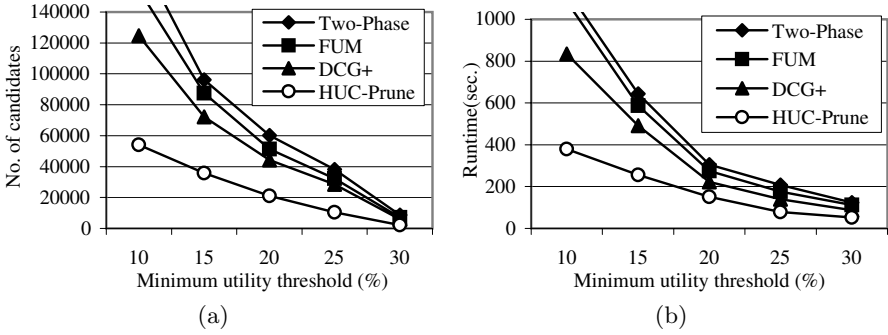


Fig. 4. Comparison on the mushroom dataset (a) Number of candidates (b) Runtime

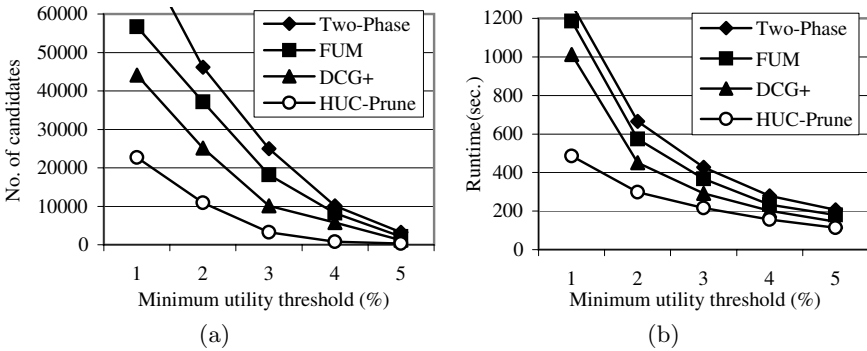


Fig. 5. Comparison on the T10I4D100K dataset (a) Number of candidates (b) Runtime

Mushroom (8,124 transactions, 119 distinct items) is a dense dataset having transaction length 23 for its every transaction. Almost 20% ($(23/119) \times 100$) of its total items are present in every transaction. Dense datasets have too many long frequent as well as high utility patterns. Because of the probability of an item’s occurrence is very high in every transaction, the number of candidate patterns and the maximum length of the candidate patterns sharply increase when the minimum threshold decreases in dense datasets. The number of candidates comparison in mushroom dataset is shown in Fig. 4(a). The number of candidates rapidly increases below the utility threshold 20%. For utility threshold 10% and 15% its amount is remarkable larger from our candidate patterns. Fig. 4(b) shows the running time comparison in mushroom dataset. As lower threshold has too many long candidate patterns and several database scans are needed for the huge number of long candidate patterns, time difference between existing algorithms and our technique becomes larger as the δ decreases. So, it is obvious that our technique is better than the existing algorithms in dense datasets.

T10I4D100K (100,000 transactions, 870 distinct items) is a sparse dataset containing average transaction length 10. Sparse datasets normally have too many distinct items. Although in the average case their transactions length is

small, but they normally have many transactions. Handling too many distinct items is a serious problem for level-wise candidate generation-and-test methodology. Therefore, scanning sparse datasets with many candidates several times is a severe problem of the existing algorithms. The number of candidate patterns and runtime comparison in this dataset are shown in Fig. 5(a) and Fig. 5(b) respectively. These figures demonstrate that our technique also outperforms the existing algorithms in sparse datasets.

6 Conclusions

The key contribution of this paper is to provide a very efficient tree-based candidate pruning technique for high utility pattern mining. Our technique prunes a huge number of candidates during tree creation time by eliminating non-candidate single-element patterns and also during mining time by using a pattern growth approach. Its maximum number of database scans is totally independent of the maximum length of candidate patterns. It needs maximum three database scans in contrast to several database scans needed for the existing algorithms. Extensive performance analyses show that our technique is very efficient in high utility pattern mining and it outperforms the existing algorithms in both dense and sparse datasets.

References

1. Agrawal, R., Srikant, R.: Fast Algorithms for Mining Association Rules. In: 20th Int. Conf. on Very Large Data Bases (VLDB), pp. 487–499 (1994)
2. Yao, H., Hamilton, H.J., Butz, C.J.: A Foundational Approach to Mining Itemset Utilities from Databases. In: Third SIAM Int. Conf. on Data Mining, pp. 482–486 (2004)
3. Yao, H., Hamilton, H.J.: Mining itemset utilities from transaction databases. *Data & Knowledge Engineering* 59, 603–626 (2006)
4. Liu, Y., Liao, W.-K., Choudhary, A.: A Two Phase algorithm for fast discovery of High Utility of Itemsets. In: Ho, T.-B., Cheung, D., Liu, H. (eds.) PAKDD 2005. LNCS(LNAI), vol. 3518, pp. 689–695. Springer, Heidelberg (2005)
5. Han, J., Pei, J., Yin, Y., Mao, R.: Mining frequent patterns without candidate generation: a frequent-pattern tree approach. *Data Mining and Knowledge Discovery* 8, 53–87 (2004)
6. Erwin, A., Gopalan, R.P., Achuthan, N.R.: CTU-Mine: An Efficient High Utility Itemset Mining Algorithm Using the Pattern Growth Approach. In: 7th IEEE Int. Conf. on Computer and Information Technology (CIT 2007), pp. 71–76 (2007)
7. Li, Y.-C., Yeh, J.-S., Chang, C.-C.: Isolated items discarding strategy for discovering high utility itemsets. *Data & Knowledge Engineering* 64, 198–217 (2008)
8. Tanbeer, S.K., Ahmed, C.F., Jeong, B.-S., Lee, Y.-K.: CP-tree: A tree structure for single pass frequent pattern mining. In: Washio, T., Suzuki, E., Ting, K.M., Inokuchi, A. (eds.) PAKDD 2008. LNCS(LNAI), vol. 5012, pp. 1022–1027. Springer, Heidelberg (2008)