Mining High Utility Patterns in Incremental Databases

Chowdhury Farhan Ahmed, Syed Khairuzzaman Tanbeer, Byeong-Soo Jeong,
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
Frequent pattern mining techniques treat all items in the database equally by taking into consideration only the presence of an item within a transaction. However, the customer may purchase more than one of the same item, and the unit price may vary among items. High utility pattern mining approaches have been proposed to overcome this problem. As a result, it becomes a very important research issue in data mining and knowledge discovery. On the other hand, incremental and interactive data mining provides the ability to use previous data structures and mining results in order to reduce unnecessary calculations when the database is updated, or when the minimum threshold is changed. In this paper, we propose a tree structure, called IIUT (incremental and interactive utility tree), to solve these problems together. It uses a pattern growth mining approach to avoid the level-wise candidate set generation-and-test problem, and it can efficiently capture the incremental data without any restructuring operation. Moreover, IIUT has the *"build once mine many"* property and therefore it is highly suitable for interactive mining. Experimental results show that our tree structure is very efficient and scalable for incremental and interactive high utility pattern mining.

Categories and Subject Descriptors
H.2.8 [**Database Applications**]: Data Mining.

General Terms
Algorithms, Management, Performance, Design, Experimentation.

Keywords
Data mining, Knowledge discovery, Frequent pattern mining, High utility pattern mining, Incremental mining, Interactive mining.

ICUIMC-09, January 15-16, 2009, Suwon, S. Korea

1. INTRODUCTION
Frequent pattern mining [1, 2, 11, 12, 13, 24] 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. However, the practical usefulness of frequent patterns is limited by the significance of the discovered patterns. There are two principal limitations: first, all items are treated as if they have the same importance/weight/price, and second, in one transaction each item appears in binary (0/1) form, i.e., as 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. As a result, frequent patterns only reflect the statistical correlation between items, and it does not reflect the semantic significance of the items.

The high utility mining [4, 5, 6, 8, 20, 21, 23] model was defined to solve the above limitations of frequent pattern mining by allowing the user to conveniently measure the importance of an itemset by its utility value. With utility mining, several important decisions in business area like maximizing revenue, minimizing marketing or inventory costs can be made and more important knowledge about itemsets/customers contributing to the majority of the profit can be discovered. In addition to real world retail markets, we can also consider biological gene databases and web click streams, where the importance of each gene or website is different and their occurrences are not limited to a 0/1 value. These techniques can also be applied to many other areas, including stock tickers, network traffic measurements, web-server logs, data feeds from sensor networks and telecom call records.

On the other hand, by using incremental and interactive pattern mining, we can use the previous data structures and mining results and avoid unnecessary calculations when the database is updated or the mining threshold is changed. However, the previous high utility pattern mining techniques [4, 5, 6, 8, 20, 21, 23] are based on a fixed database and so they do not consider that one or more transactions may be deleted, inserted or modified in the database. To understand the necessity of today's incremental databases, where additions, deletions, and modifications are frequent operations, we can consider a small example in the real world market. Say customer *X* has bought 3 pens, 4 pencils and 1 eraser and customer *Y* has bought 1 computer mouse. After some time,

customer *Z* may come to buy 2 loaves of bread and 1 carton of milk, customer *X* may return 2 pencils, and customer *Y* may return the mouse. As one can see, in the real world, new transactions can be added, and old transactions may be frequently modified or deleted. As a result, we must consider additions, deletions and modifications to the transactions in real world datasets.

Moreover, the existing data structures do not have the *"build once mine many"* property (by building the data structure only once, several mining operations can be done) for interactive mining. As a result, they cannot use their previous data structures and mining results for the new mining threshold. In our real world, however, the users need to repeatedly change the minimum threshold for useful information extraction according to their application requirements. Therefore, the *"build once mine many"* property is essentially needed to solve these interactive mining problems.

Motivated by these real world scenarios, we propose a novel tree structure, called IIUT (Incremental and Interactive Utility Tree), for high utility pattern mining in incremental databases. It exploits a pattern growth mining approach to avoid the level-wise candidate generation-and-test problem of the existing high utility pattern mining algorithms [4, 5, 6, 8, 23]. It needs only two database scans, in contrast to several database scans needed for the existing algorithms. We design the IIUT structure according to the item's appearance order, and it can efficiently handle the incremental data without any restructuring operations. Moreover, IIUT has the *"build once mine many"* property and therefore it is very efficient in interactive mining. Extensive experimental results show that our approach is very efficient for incremental and interactive 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 the construction, incremental maintenance and mining operation of our proposed tree structure IIUT for high utility pattern mining. In Section 5, our experimental results are presented and analyzed. Finally, in Section 6, conclusions are drawn.

2. RELATED WORK

In the following subsections we discuss previous research on frequent pattern mining, weighted frequent pattern mining, high utility pattern mining, incremental and interactive pattern mining.

2.1 Frequent Pattern Mining

Let $I = \{i_1, i_2, \ldots\ldots i_m\}$ be a set of items and D be a transaction database $\{T_1, T_2, \ldots\ldots T_n\}$ where each transaction $T_i \in D$ is a subset of I. The support/frequency of a pattern $X\{x_1, x_2, \ldots\ldots x_p\}$ is the number of transactions containing the pattern in the transaction database. The problem of frequent pattern mining is to find the complete set of patterns satisfying a minimum support in the transaction database. The *anti-monotone* property [1, 2] is used to prune the infrequent patterns. This property tells that if a pattern is infrequent then all of its super patterns must be infrequent. *Apriori* [1, 2] algorithm is the initial solution of the frequent pattern mining problem. But it suffers from the level-wise candidate generation-and-test problem and needs several database scans. The FP-growth [11] algorithm solved this problem by using an FP-tree based solution without any candidate generation and using only two database scans. Other research [12, 13, 18, 24] has also been done in this area. This traditional frequent pattern mining considers equal profit/weight for all items and only binary occurrence (0/1) of the items in one transaction.

2.2 Weighted Frequent Pattern Mining

Research has been done for weighted frequent pattern mining [3, 9, 10, 19] in binary database where in each transaction frequency of one item can be either 1 or 0. These techniques showed that the main challenging problem in this area is that the itemset weighted frequency does not have the *anti-monotone* property. Weight of a pattern P is the ratio of sum of all its items weight value and length of P. Consider item *"a"* has weight 0.6 and frequency 4, item *"b"* has weight 0.2 and frequency 5, itemset *"ab"* has frequency 3. Then the weight of itemset *"ab"* will be $(0.6 + 0.2)/2 = 0.4$ and its weighted frequency will be $0.4 \times 3 = 1.2$. Weighted frequency of *"a"* is $0.6 \times 4 = 2.4$ and "b" is $0.2 \times 5 = 1.0$. If the minimum weighted frequency threshold is 1.2, then pattern *"b"* is weighted infrequent but *"ab"* is weighted frequent. They maintain the *anti-monotone* property by multiplying each itemset's frequency by the global maximum weight. In the above example, if *"a"* has the maximum weight of 0.6, then by multiplying it with the frequency of item *"b"*, 3.0 is obtained. Hence, pattern *"b"* is not pruned at the early stage and pattern *"ab"* will not be missed, but pattern *"b"* is overestimated and will be pruned later by using its actual weighted frequency. Maintaining the *anti-monotone* property in high utility pattern mining area is more challenging, as it considers non-binary frequency values of items in each transaction.

2.3 High Utility Pattern Mining

The Itemset Share approach [7] considers non-binary frequency of an item in each transaction. Share is percentage of a numerical total that is contributed by the items in an itemset. They define the problem of finding share frequent itemsets and compared the share and support measures to illustrate that share measure can provide useful information about numerical values that are associated with transaction items, which is not possible using only the support measure. They cannot rely on the *anti-monotone* property [1, 2]. They developed heuristic methods to find itemsets with share values above the minimum share threshold. Mining high utility itemsets [8] developed top-K objective-directed high utility closed patterns. Their definitions are different from our work. They assume the same medical treatment for different patients (different transactions) will have different levels of effectiveness. Furthermore, a pruning strategy was developed to prune low utility itemsets based on a weaker anti-monotone condition.

The theoretical model and definitions of high utility pattern mining were given in [4]. This approach, called MEU (mining with expected utility), cannot maintain the *anti-monotone* 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

TID \ ITEM	a	b	c	d	e	Trans. Utility($)
T_1	0	2	8	2	0	52
T_2	4	8	0	0	0	56
T_3	0	3	6	0	4	76
T_4	2	2	7	0	0	37
T_5	2	0	8	0	0	28
T_6	6	5	0	4	0	74

(a) Transaction Database

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

(b) Utility Table

Figure 1. Example of a transaction database and utility table

same authors proposed two new algorithms UMining and UMining_H [5] to calculate the high utility patterns. In UMining they have used a pruning strategy based on the 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 this heuristic method, and these methods once again do not satisfy the *anti-monotone* property of *Apriori,* leading to an overestimation of the number of patterns. They also suffer from the level-wise candidate generation-and-test problem.

The Two-Phase [6] algorithm was developed based on the definitions of [4] to find high utility itemsets using the anti-monotone property of *Apriori*. Here, the authors define the *"transaction weighted utilization"* and prove it can maintain the *anti-monotone* property. In the first database scan, their algorithm finds all the 1-element transaction weighted utilization itemsets and, based on that, generates the candidates for 2-element transaction weighted utilization itemsets. In the second database scan, the algorithm finds all the 2-element transaction weighted utilization itemsets and uses this information to generate the candidates for the 3-element transaction weighted utilization itemsets, and so on. At the last scan, the algorithm discovers the actual high utility itemsets from the high transaction weighted utilization itemsets. However, this algorithm suffers from the same problems of level-wise candidate generation-and-test methodology. CTU-Mine [20] is an algorithm that is more efficient than Two-Phase only in dense databases, when the minimum utility threshold is very low. In [21], another algorithm is presented that uses an approximation to solve the high utility pattern mining problem through specialized partition trees, called high-yield partition trees. The isolated items discarding strategy (IIDS) [23] for discovering high utility itemsets was proposed to reduce some candidates in every pass of the databases. It shows that the utility mining problem can be directly converted from the itemset share mining problem [7]. Applying IIDS, the authors developed an efficient high utility itemset mining algorithm FUM and DCG+ and showed that their work is better than all previous high utility pattern mining algorithms. However, their algorithms still suffer from the level-wise candidate set generation-and-test problem and need multiple database scans.

2.4 Incremental and Interactive Mining

Research has been done [14, 15, 16, 18, 24] to develop incremental and interactive mining for traditional frequent pattern mining, showing that incremental prefix tree structures are quite possible and efficient using currently available memory sizes in the gigabyte range. The efficient dynamic database updating algorithm (EDUA) [22] was designed for mining databases when data deletion is carried out frequently in any subset of a database. The IncWTP and WssWTP algorithms [17] are designed for incremental and interactive mining of web traversal patterns.

In the existing work on high utility pattern mining, no one has proposed any solution for incremental mining, where many new transactions can be added and existing transactions can be deleted or modified. Moreover, none of the data structures have the *"build once, mine many"* property. This leads us to propose a new tree structure for incremental high utility pattern mining with the *"build once, mine many"* property that also uses a pattern growth mining operation to avoid the level-wise candidate set generation-and-test problem.

3. PROBLEM DEFINITION

We have adopted similar definitions presented in the previous works [4-6]. Let $I = \{i_1, i_2, \ldots\ldots i_m\}$ be a set of items and D be a transaction database $\{T_1, T_2, \ldots\ldots T_n\}$ where each transaction $T_i \in D$ is a subset of I.

Definition 1: The local transaction utility value $l(i_p, T_q)$, represents the quantity of i_p in transaction T_q. For example, in Figure 1(a) $l(b, T_2) = 8$.

Definition 2: The external utility $p(i_p)$ is the unit profit value of item i_p. For example, in Figure 1(b), $p(b) = 6$.

Definition 3: Utility $u(i_p, T_q)$, is the quantitative measure of utility for item i_p in transaction T_q, defined by

$$u(i_p, T_q) = l(i_p, T_q) \times p(i_p) \quad (1)$$

For example, $u(b, T_2) = 8 \times 6 = 48$ in Figure 1.

Definition 4: The utility of an itemset X in transaction T_q, $u(X, T_q)$ is defined by,

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

where $X = \{i_1, i_2, \ldots\ldots i_k\}$ 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 Figure 1.

Definition 5: The utility of an itemset X is defined by,

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

For example, $u(ab) = u(ab, T_2) + u(ab, T_4) + u(ab, T_6) = 56 + 16 + 42 = 114$ in Figure 1.

Definition 6: The transaction utility of transaction T_q denoted as $tu(T_q)$ means the total profit of that transaction and it is defined by,

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

For example, $tu(T_1) = u(b, T_1) + u(c, T_1) + u(d, T_1) = 12 + 24 + 16 = 52$ in Figure 1.

Definition 7: The minimum utility threshold δ, is given by percentage of the total transaction utility values of the database. In Figure 1, the summation of all the transaction utility values is 323. If δ is 25% or we can also express it as *0.25*, then minimum utility value can be defined as

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

So, in this example $minutil = 0.25 \times 323 = 80.75$ in Figure 1.

Definition 8: An itemset X is a high utility itemset, if $u(X) \geq minutil$. Finding high utility itemsets means find out all the itemsets X having $u(X) \geq minutil$. For $munutil = 80.75$, pattern *"ab"* is a high utility pattern, as $u(ab) = 114$ (calculated in the example of Definition 5)

The main challenging problem of high utility pattern mining area is itemset utility does not have the *anti-monotone* property. For example, if $minutil = 80.75$ in Figure 1, then "*d"* is a low utility item as $u(D) = 48$, but *"bd"* is a high utility itemset as $u(bd) = 90$ according to equation (3) and Definition 8. According to Definition 5 and equation (3), the definition of itemset's utility is based on the actual sum of product calculation similar to our real world business calculations. Maintaining the *anti-monotone* property is more challenging here compared to the weighted frequent itemset mining problem for two reasons. First, weighted frequent itemset mining considers the 0/1 appearance of an item inside a transaction, and second, the weight of an itemset is calculated using the average of all the items' weights inside of it.

In high utility pattern mining we can maintain the *anti-monotone* property by transaction weighted utilization.

Definition 9: The transaction weighted utilization of an itemset X, denoted by $twu(X)$, is the sum of the transaction utilities of all the transactions containing X.

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

For example, $twu(ac) = tu(T_4) + tu(T_5) = 37 + 28 = 65$ in Figure 1. The *anti-monotone* property can be maintained using transaction weighted utilization. Here, for $minutil = 80.75$ in Figure 1, as $twu(ac) < minutil$, any super pattern of *"ac"* cannot be high *twu* itemset and obviously can not be high utility itemset.

Definition 10: X is a high transaction weighted utilization itemset if $twu(X) \geq minutil$.

In our algorithm, after finding all the high *twu* patterns maintaining the *anti-monotone* property using only one database scan, we calculate all the high utility patterns from the high *twu* patterns by scanning the database a second time and performing the original utility calculations (according to equation 3) for those high *twu* patterns.

4. OUR PROPOSED TREE STRUCTURE

In this section, at first we describe the construction and incremental maintenance process of IIUT (Incremental and Incremental Utility Tree). After that, we show the mining process of IIUT using a pattern growth approach.

4.1 Construction and Maintenance

In the first database scan IIUT arranges the items inside a transaction in appearance order and inserts as a branch inside the tree. Header table is also maintained like FP-tree [11]. IIUT explicitly maintains *twu* (transaction weighted utilization) value in both header table and inside the nodes of the tree. To facilitate the tree traversals, adjacent links are also maintained (not shown in the figures for simplicity).

Consider the transaction database and utility table in Figure 1. The database contains 6 transactions. The construction process of IIUT is described in Figure 2. The *tu* value of T_1 is 52 in Figure 1. Figure 2(a) shows that T_1 is added in the tree. The first node in the appearance order is *"b"* with a *twu* value of 52. The second node and third node is *"c"* and *"d"* respectively with the same *twu* value. Figure 2(b) shows the IIUT after addition of T_2. As shown in the header table of Figure 2(a), three items are already present there. Hence, according to appearance order, item *"a"* is the fourth item. Arranging items in their appearance order, we get the advantage that we do not have to rearrange the header table or the tree. For the new transactions, we arrange the items of the

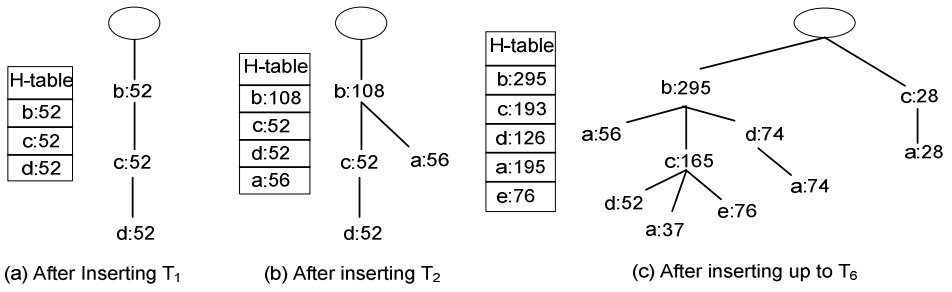

Figure 2. Construction process of IIUT

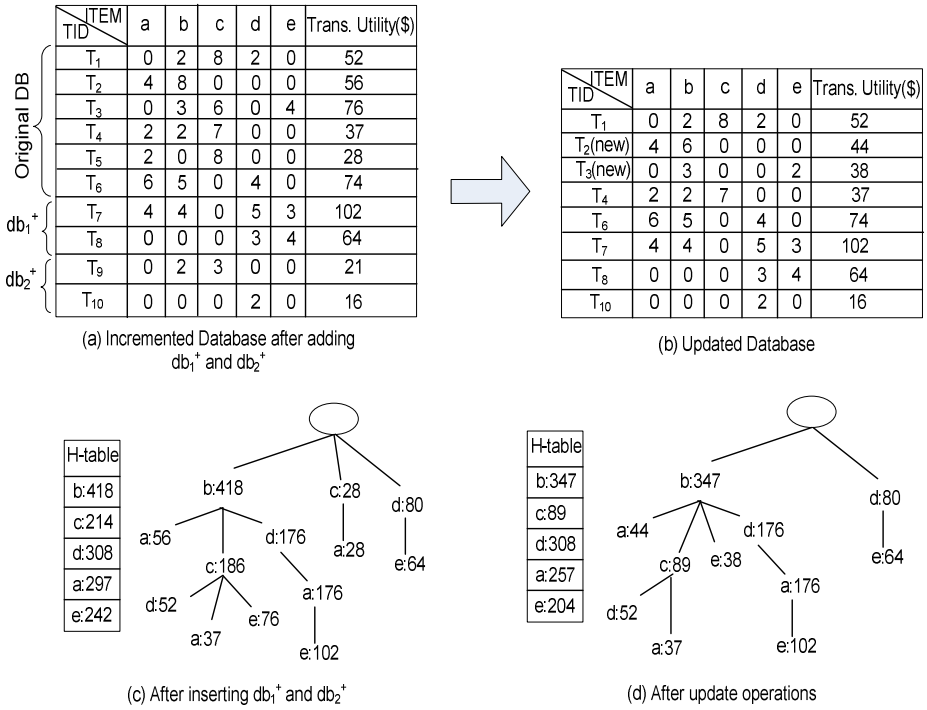

Figure 3. Incremental maintenance of IIUT

transaction according to the appearance order as maintained in the header table and insert them into the tree. If any new item comes, it is the last item of the header table. This arranges T_2 as *"b a"* with a *tu* value of 56, and so, the branch, in order of appearance, is *"b:56, a:56"*. Here *"b"* gets the prefix sharing with the existing node *"b"* and the *twu* value of node *"b"* is 52+56=108, and item *"a"* becomes its new child. All other transactions are inserted into the tree in a similar manner, as shown in Figure 2(c).

The following properties are true for IIUT.

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

Property 2: The ordering of items in IIUT is unaffected by the changes in *twu* values caused by incremental updating.

Property 3: IIUT can be constructed using a single pass of a database.

Figure 3(a) shows our initial database of Figure 1 is increased by adding two groups of transactions, db_1^+ and db_2^+. Figure 3(b) shows our current database of Figure 3(a) is updated by deleting transactions T_5 and T_9 and by modifying transaction T_2 and T_3. Figure 3(c) shows IIUT can be easily incremented to capture db_1^+ and db_2^+. The insertion process is same as transactions are inserted in the construction process.

To delete T_5, we reduce the *twu* value of T_5 (28), along the path "c a". As the *twu* values of the nodes become zero, we delete them from IIUT (shown in Figure 3(d)). Next, to delete T_9, we have to reduce the *twu* value in the branch containing *"b"* and *"c"*. After reduction, the new *twu* value of *"b"* and *"c"* items in that branch is 397 and 165 respectively (shown in Figure 3(d)). All the reduced *twu* values are also adjusted in the header table. Figure 3(d) also shows the IIUT after modification. In the first modification, T_2 is modified by decreasing the quantity value of item *"b"* by 2. So, *twu* value has to be reduced by 2×6 = 12, for that transaction. In the second modification, item *"c"* is totally removed and quantity of item *"e"* is reduced by 2 from T_3. The *twu* value of item *"c"* has to be reduced by *twu* value of T_3 as item *"c"* is no longer a part of this transaction. The *twu* values of the other two items *"b"* and *"e"* will be reduced by 6×3 = 18 for the 6 removed items *"c"* and 2×10 = 20 for the 2 removed items *"e"*. So, in total, the *twu* value has to be reduced by 18+20=38 for items *"b"* and *"e"*, and node *"e"* now becomes a child of node *"b"* after the modification.

4.2 Mining Process

As our IIUT has the important property of FP-tree stated in property 1, pattern growth mining approach can be directly applicable to IIUT by using *twu* values. Consider the last updated database of Figure 3(b) and δ = 0.25. According to equation 5, *minutil* = 106.75. As in pattern growth, we start from the bottom-most item.

At first the conditional tree of the bottom-most item *"e"* (shown in Figure 4(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

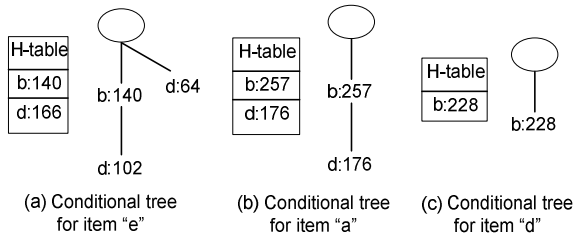

Figure 4. Mining process of IIUT

for item *"a"* is created in Figure 4(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 Figure 4(c) and candidate patterns (8) {*b, d:* 228}, (9) {*d:* 308} are generated. As item *"c"* has low *twu* value 89, we can discard it without further calculation. The last candidate pattern (10) {*b:* 347} is generated for the top-most item *"b"*. Second database scan is required to find high utility patterns from these ten high *twu* patterns. The six high utility patterns are {*a, b:* 134}, {*a, b, d:* 146}, {*b:* 132}, {*b, d:* 154}, {*d:* 128} and {*d, e:* 134}.

4.3 Interactive Mining

IIUT has the *"build once, mine many"* property, meaning that after creating one tree, several mining operations can be performed using different minimum thresholds. For example, after the creation of IIUT in Figure 3(d), we first set δ = 0.25 (25%), but then, after finding the results, we can set other δ values, such as 0.2, 0.3, 0.1, 0.4, and so on. After the first mining threshold, we do not have to create the tree. There are also advantages in mining time and second database scan time for finding high utility patterns from high *twu* patterns. For example, after getting 6 high utility patterns from 10 candidate *twu* patterns for δ = 0.25, if we then use larger values of δ, like 0.3, the candidates are subsets of the previous candidates and we can determine which are actual high utility itemsets without mining or scanning the database again. If the new δ is smaller than the previous one, then the candidate set is a super set to the previous candidate set, and so, after mining, we have to scan the database only for those patterns that did not appear in the previous candidate set.

5. EXPERIMENTAL RESULTS

To evaluate the performance of our proposed tree structure, we have performed several experiments on the IBM synthetic T10I4D100K dataset, and the real-life mushroom and kosarak datasets from frequent itemset mining dataset repository (http://fimi.cs.helsinki.fi/data/) and UCI Machine Learning Repository (http://kdd.ics.uci.edu/). These datasets do not provide the profit values or quantity of each item for each transaction. As like the performance evaluation of the previous weight/utility based pattern mining [3, 6, 9, 10, 19, 20] 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 [6, 23]. 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.

To show the power of the *"build once, mine many"* property in interactive mining, as well as the effects of only two database scans and the very reduced candidate sets obtained by pattern growth mining, we compare our IIUT with the existing Two-Phase, FUM and DCG+ algorithms using both dense and sparse datasets.

5.1 Effectiveness in Interactive Mining

Mushroom (0.56 MB, 8,124 transactions, 119 distinct items) is a dense dataset having transaction length of 23 for its every transaction. Around 20% of its total items are present in every transaction. So, it has many candidate patterns of long length. We first compared the number of candidates, shown in Figure 5(a). Figure 5(b) shows the running time comparison in the mushroom dataset for the worst case of interactive mining. As we stated in interactive mining in section 4.3, our IIUT gets benefit after the first mining threshold.

We have taken here the result in Figure 5(b) for the worst case of our tree, i.e., we have given the thresholds in descending order (at first 30% then 25% and so on). After the first threshold our trees do not have to be constructed. As the threshold decreases, mining

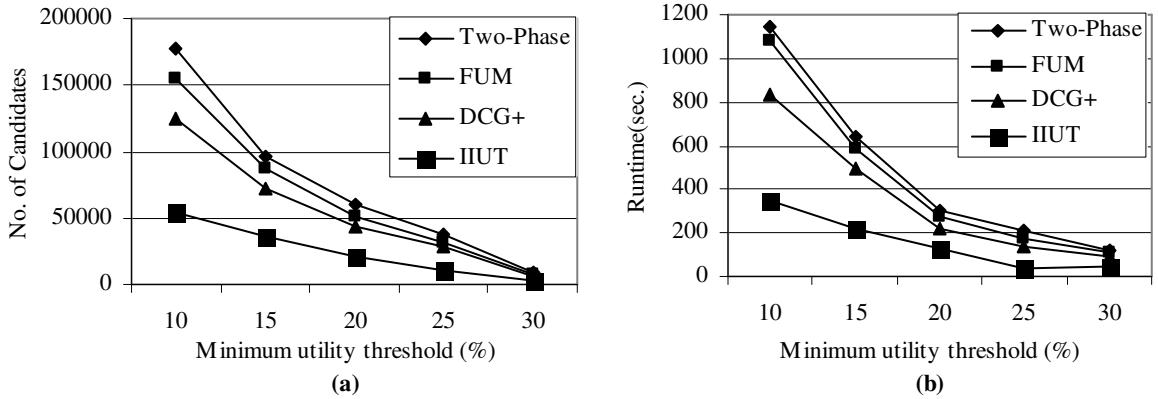

Figure 5. Performance comparison on the mushroom dataset, (a) No. of candidates (b) Runtime

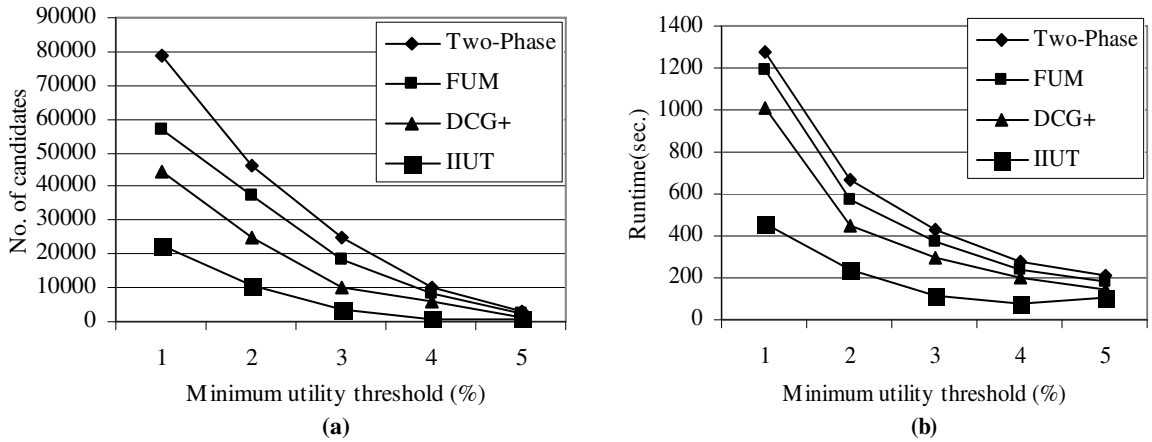

Figure 6. Performance comparison on the T10I4D100K dataset, (a) No. of candidates (b) Runtime

operation is needed. As candidate set of threshold 25% is a superset of the candidate set of threshold 30%, we can easily reduce the second database scan time by using the candidate set and resultant high utility patterns already calculated for threshold 30%. We have to scan the database second time only for those candidates newly added for threshold 25%. In the same way, we calculate the high utility patterns for the other thresholds.

The best case occurs if we go for mining threshold in increasing order, i.e., at first 10% then 20% and so on. Then candidate set of threshold 20% is a subset of the candidate set of threshold 10%, and therefore, we can find the resultant high utility patterns from previous results, without mining or a second database scan. In that case, after the first mining threshold, the computation times for the other thresholds are almost negligible compared to the first one. The running time difference between existing level-wise *Apriori* algorithms and our algorithm becomes larger with as the δ decreases because of several database scans with too many candidates.

T10I4D100K (4 MB, 100,000 transactions, 870 distinct items) is a sparse dataset containing average transaction length of 10. The number of candidates comparison is shown in Figure 6(a). The result of running time in this dataset is shown in Figure 6(b). It also reflects our analysis like the dense mushroom dataset. For the worst case analysis in Figure 6(b), we have also given the value of δ here in descending order (starting with 5%). As discussed in the analysis for the mushroom dataset, we do not need to construct the tree after first threshold and also we can get some pre-calculated candidate patterns and their resultant high utility patterns form the previous result.

5.2 Effectiveness in Incremental Mining

We have tested the effectiveness of IIUT in incremental mining with the kosarak dataset (30.5 MB). It has almost 1 million transactions (990,002) and 41,270 distinct items. At first we have created the IIUT for 0.2 million transactions of this dataset and perform mining operation with the minimum threshold 7%. Then we perform the interactive mining on that tree with other three thresholds 6%, 5% and 4%. Another 0.2 million transactions were added to the tree and the mining operations performed with the same minimum thresholds and in a similar order. The rest of the transactions in the kosarak dataset were added in a similar manner and the mining operations were performed at each stage. The results are shown in Figure 7, where it is obvious that as the database increases, the tree construction and mining time also increase, and, in each stage, the tree construction cost is only needed for the minimum threshold 7%.

IIUT efficiently handled 41270 distinct items in the kosarak dataset. In Figure 7, the trees containing 0.2 million, 0.4 million, 0.6 million, 0.8 million and 1 million transactions have 28780, 34062, 37030, 39562 and 41270 distinct items, respectively. In other words, Figure 7 shows the scalability of IIUT in handling large numbers of distinct items and transactions.

5.3 Memory Usage

Previous prefix tree based frequent mining research [14, 15, 16, 18, 24] has shown that the memory requirement for the prefix trees is feasible and efficient by using the gigabyte-range memory now available. We have also handled our IIUT very efficiently within this memory range. The IIUT constructed for the full mushroom, T10I4D100K and kosarak datasets required 0.836

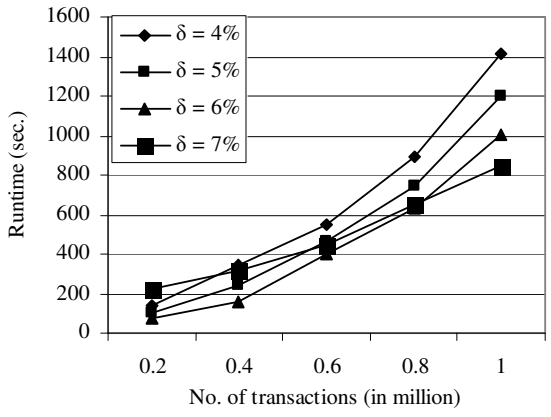

Figure 7. Dataset increasing by db^+ = 0.2 million transactions on the kosarak dataset

MB, 16.697MB and 219.784MB of memory, respectively, and so, using our IIUT prefix-tree structure for high utility pattern mining is not only quite feasible, but also efficient.

6. CONCLUSIONS

We have proposed an efficient tree structure called IIUT that adjusts itself dynamically for incremental and interactive high utility pattern mining without any restructuring operations. IIUT has the *"build once, mine many"* property and is highly suitable for interactive mining. It reduces considerably the mining time required by using previous tree structures and mining results. Moreover, it exploits a pattern growth mining approach to avoid the level-wise candidate generation-and-test problem, and uses a maximum of two database scans. Extensive performance analyses show that our tree structure is very efficient and scalable in incremental and interactive high utility pattern mining.

7. ACKNOWLEDGMENTS

This study was supported by a grant of the Korea Health 21 R&D Project, Ministry for Health Welfare and Family Affairs, Republic of Korea (A020602).

8. REFERENCES

[1] Agrawal, R., Imieliński, T., and Swami, A. Mining association rules between sets of items in large databases. In ACM SIGMOD Int. Conf. on Management of Data, pp. 207-216, 1993.

[2] Agrawal, R., and Srikant, R. Fast Algorithms for Mining Association Rules. In 20th Int. Conf. on Very Large Data Bases, pp. 487-499, 1994.

[3] Yun, U. Efficient Mining of weighted interesting patterns with a strong weight and/or support affinity. Information Sciences, vol. 177, pp. 3477-3499, 2007.

[4] Yao, H., Hamilton, H. J., and C. J. Butz. A Foundational Approach to Mining Itemset Utilities from Databases, In Proceedings of the Third SIAM International Conference on Data Mining, pp. 482-486, 2004.

[5] Yao, H., and Hamilton, H. J. Mining itemset utilities from transaction databases, Data &Knowledge Engineering, vol. 59, pp. 603-626, 2006.

[6] Liu, Y., Liao, W.-K., and Choudhary, A. A Two Phase algorithm for fast discovery of High Utility of Itemsets, In Proceedings of the 9th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'05), pp. 689-695. 2005.

[7] Barber, B., and Hamilton, H. J. Extracting share frequent itemsets with infrequent subsets, Data Mining and Knowledge Discovery, vol. 7, pp. 153-185, 2003.

[8] Chan, R., Yang, Q., and Shen, Y. D. Mining High Utility Itemsets, In Proceedings of the Third IEEE International Conference on Data Mining, pp. 19-26, 2003.

[9] Tao, F. Weighted association rule mining using weighted support and significant framework. In 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, pp. 661–666, 2003.

[10] Wang, W., Yang, J., and Yu, P. S. WAR: weighted association rules for item intensities. Knowledge Information and Systems, vol. 6, pp. 203–229, 2004.

[11] Han, J., Pei, J., Yin, Y., and Mao, R. Mining frequent patterns without candidate generation: a frequent-pattern tree approach. Data Mining and Knowledge Discovery, vol. 8, pp. 53–87, 2004.

[12] Grahne, G., and Zhu, J. Fast Algorithms for frequent itemset mining using FP-Trees. IEEE Transactions on Knowledge and Data Engineering, vol. 17, no. 10, pp. 1347 – 1362, 2005.

[13] Han, J., Cheng, H., Xin, D., and Yan, X. Frequent pattern mining: current status and future directions. Data Mining and Knowledge Discovery, vol. 15, pp. 55–86, 2007.

[14] Cheung, W., and Zaïane, O. R. Incremental Mining of Frequent Patterns Without Candidate Generation or Support Constraint, In Proceedings of the Seventh International Database Engineering and Applications Symposium (IDEAS'03), pp.111-116, 2003.

[15] Koh, J.-L., Shieh, S.-F. An Efficient Approach for Maintaining Association Rules Based on Adjusting FP-tree Structures. In Proceedings of DASFAA 2004. LNCS, vol. 2973, pp. 417–424, 2004.

[16] Li, X., Deng, Z. -H., and Tang, S. A fast algorithm for maintenance of association rules in incremental databases. Advanced Data Mining and Applications (ADMA), vol. 4093, pp. 56-63, 2006.

[17] Lee, Y. -S., Yen, and S. –J. Incremental and interactive mining of web traversal patterns, Information Science, vol. 178, pp. 287-306, 2008.

[18] Leung, C. K. -S., Khan, Q. I., Li, Z., and Hoque, T. CanTree: a canonical-order tree for incremental frequent-pattern mining. Knowledge and Information Systems, vol. 11, no. 3, pp. 287-311, 2007.

[19] Yun, U. Mining lossless closed frequent patterns with weight constraints. Knowledge-Based Systems, vol. 210, pp. 86–97, 2007.

[20] Erwin, A., Gopalan, R. P., and Achuthan, N.R. CTU-Mine: An Efficient High Utility Itemset Mining Algorithm Using the Pattern Growth Approach. In Proceedings of the Seventh IEEE International Conference on Computer and Information Technology (CIT'07), pp. 71–76, 2007.

[21] Hu, J., and Mojsilovic, A. High Utility Pattern Mining: A method for discovery of high utility item sets, Pattern Recognition, vol. 40, pp. 3317-3324, 2007.

[22] Zhang, S., Zhang, J., and Zhang, C. EDUA: An efficient algorithm for dynamic database mining, Information Sciences, vol. 177, pp. 2756-2767, 2007.

[23] Li, Y. –C., Yeh, J. –S., and Chang, C. –C. Isolated items discarding strategy for discovering high utility itemsets, Data & Knowledge Engineering 64, pp. 198-217, 2008.

[24] Tanbeer, S. K., Ahmed, C. F., Jeong, B. -S., Lee, Y. -K. CP-tree: A tree structure for single pass frequent pattern mining. In: 12th Pacific-Asia Conf. on Knowledge Discovery and Data Mining (PAKDD), pp. 1022-1027, 2008.