Efficient Algorithms for Incremental Utility Mining

Jieh-Shan Yeh
Department of Computer Science and Information Management
Providence University
Taichung 433, Taiwan
+886-4-26328001
jsyeh@pu.edu.tw

Chih-Yang Chang
Department of Computer Science and Information Management
Providence University
Taichung 433, Taiwan
+886-4-26328001
g9471069@pu.edu.tw

Yao-Te Wang
Department of Computer Science and Information Management
Providence University
Taichung 433, Taiwan
+886-4-26328001
ytwang@pu.edu.tw

ABSTRACT
Temporal data mining is the activity of finding interesting correlations or patterns in large temporal data sets. On the other hand, utility mining aims at identifying the itemsets with high utilities. In 2006, Tseng et al. introduced the temporal utility mining which is extended from both temporal association rule mining and utility mining. In this study, we investigated the incremental utility mining which can identify all high temporal utility itemsets in a specified time period on an incremental transaction database. Two efficient algorithms, Incremental Utility Mining (IUM) and Fast Incremental Utility Mining (FIUM), were proposed. The experimental results also showed that both algorithms are efficient.

Categories and Subject Descriptors
H.2.8 [**Database Management**]: Database Applications – *data mining.*

General Terms
Algorithms, Performance, Experimentation, Theory.

Keywords
Temporal data mining, incremental mining, utility mining.

1. INTRODUCTION
In traditional association rule mining model, the products of transactions are treated as items. Association rule mining identifies relationship among a set of items frequently purchased together. Apriori [1] is the most famous algorithm for finding association rules. Some researchers continued to improve it afterward, such as [2], [3], [4], [5], [6].

However, in many applications, mining temporal association patterns from the most recent data is also important. The concept of incremental mining was introduced by Cheung et al. [7] in 1996. FUP Algorithm was proposed to solve such problem. Afterward, there were many researches on incremental sequential pattern mining aimed at the cases of incremental databases [8], [9], [10], [11], [12], [13].

Utility mining [14] is the research which discusses the quantity of items. Utility is defined as how useful or valuable an item is. Utility mining identifies high utility itemsets that own a large portion of the total utility in the transaction database. By combining the concept of temporal data mining and utility mining, Tseng et al. [15] introduced THUI-Mine algorithm for finding temporal high utility itemsets from data streams.

In this paper, we investigated the incremental utility mining which can identify all high temporal utility itemsets in a specified time period on an incremental transaction database. This study proposed two novel algorithms, Incremental Utility Mining (IUM) Algorithm and Fast Incremental Utility Mining (FIUM) Algorithm, which can discover all high temporal utility itemsets.

The rest of this paper is organized as follows. Section 2 overviews the related works. Section 3 describes the proposed algorithms. Performance studies are given in Section 4. Finally, we conclude in Section 5 with a summary of our work.

2. RELATED WORKS
2.1 Utility Mining
Let $\boldsymbol{I} = \{i_1, i_2, \ldots, i_n\}$ be a set of all items, $\boldsymbol{DB}$ be a transaction database. The external utility of item $\boldsymbol{i_k} \in \boldsymbol{I}$, $\boldsymbol{s(i_k)}$, is the value associated with item $\boldsymbol{i_k}$ in the utility table. The external utility reflects how useful an item is. For example, in Table 1, the external utility of item A, s(A), is 5.

Table 1. An example of utility table and transaction table

Item	Profit($
A	5
B	1
C	13
D	50

TID	A	B	C	D
T_1	0	0	3	5
T_2	1	2	0	0
T_3	0	3	1	0
T_4	5	0	1	0
T_5	1	0	0	2
T_6	0	0	1	2
T_7	3	1	0	1
T_8	1	0	3	3
T_9	0	5	0	0
T_{10}	0	1	1	0

The utility of item $\boldsymbol{i_k} \in \boldsymbol{I}$ in transaction $\boldsymbol{T_p}$, $\boldsymbol{u(i_k, T_p)}$, is the quantitative measure of utility for item $\boldsymbol{i_k}$ in transaction $\boldsymbol{T_p}$. For

example, in Table 2, u(A,T_4) = 5×5 = 25. Let u(X, T_p) be the utility of itemset X in transaction T_p. For example, u({ABD}, T_7) = 3×5 + 1×1 + 1×50 = 66. Let u(X) be the utility of itemset X, which is the sum of the utilities of X in all transaction containing X. For example, X = {A, B}, u(X) = u(X, T_2) + u(X, T_7) = 25 + 16 = 41.

Let u be the minimum threshold, the utility mining is to find all itemset with utility will be greater than or equal to u. However, the **downward closure property** does not hold for the utility mining. That is, a high utility itemset may contain some low utility itemsets. In Table 1, we assume that the minimum utility threshold is 100. The itemset {ACD} is a high utility itemset, because u({ACD}) = 194. The itemset {AC} is a sub-itemset of {ACD}, and u({AC}) = 82. Therefore, it is a difficult work to find efficiently all high utility itemsets, and it's also a challenge to restrict the size of the candidate set.

Although the downward closure property does not hold in the utility model. Liu et al. [16], [17] proposed the Two-Phase algorithm to set up the transaction-weight utility which can comply with the transaction-weight downward closure property. The definition as follows: The transaction utility of transaction T_i, denoted as tu(T_i), is the sum of utilities of all items in T_i. The transaction-weighted utilization of an itemset X, denoted as twu(X), is the sum of the transaction utilities of all the transactions containing X.

Table 2. Transaction utility of the transaction database

TID	Transaction	TID	Transaction
T_1	289	T_6	113
T_2	25	T_7	66
T_3	16	T_8	194
T_4	38	T_9	5
T_5	105	T_{10}	28

For example, in Table 2, twu(B) = tu(T_2) + tu(T_3) + tu(T_7) + tu(T_9) + tu(T_{10}) = 140, twu(C) = tu(T_1) + tu(T_3) + tu(T_4) + tu(T_6) + tu(T_8) + tu(T_{10}) = 678 and twu({BC}) = tu(T_3) + tu(T_{10}) = 44. If the minimum threshold is 40, itemset {BC} becomes a high transaction-weight utilization itemset. Its sub-itemset {B} and {C} are also high transaction-weight utilization itemsets.

In addition, Li et al. [18] proposed the share framework to develop FSM-algorithm which takes advantage of the level closure property to discover all high share itemsets. And then Li et al. [19] proposed an Enhanced FSM-algorithm to improve the performance of FSM-algorithm. Li et al. also proposed the ShFSM-algorithm to efficiently lower the number of useless candidates.

2.2 Temporal Data Mining

Temporal data mining searches for interesting correlations or patterns in large sets of temporal data. Chang et al. [18] introduced research for temporal association rule mining. Traditional association rule algorithms can't find the temporal association rules in the particular periods.

Temporal association rules mining doesn't consider the utility of every item. Temporal utility mining is a research which is extended from temporal association rules mining and utility mining. Tseng et al. [20] introduced an efficient algorithm, THUI-Mine, to finding temporal high utility itemsets from data streams. THUI-Mine was based on the principle of Two-Phase algorithm [21], and was extended by SWF-algorithm to be applicable in incremental database. The method can effective reduce the number of 2-itemset candidates, and used the same way to generate k-itemset (k ≥ 2) candidates.

3. PROPOSED ALGORITHMS

Let {i_1, i_2, …, i_m} be a set of all items, DB be the original transaction database segmented into n partitions {P_1, P_2, ..., P_n}. Let u(X) bs the utility of itemset X. The transaction-weighted utilization of an itemset X, denoted as twu(X), is the sum of the transaction utilities of all the transactions containing X. The high temporal utility itemsets are the itemsets with utilities greater than or equal to a specified threshold in particular periods. That is, the high temporal utility itemsets are the high utility itemsets for particular periods. The incremental utility mining is to find all high temporal utility itemsets. The definitions of other notations used in this paper are given in Table 3.

Table 3. Definitions of notations

$db^{[i,j]}$	Partition database from P_i to P_j, where i<j.
s	Utility threshold in each partition.
$LU^{[i,j]}$	High temporal utility itemsets in P_i to P_j.
RU^k	High transaction-weight utility itemsets in partition k.
C^k_p	The candidate p-itemsets in partition k.
$I.u$	The utility of itemset I.
$I.twu$	The transaction utility of itemset I.
$pcoun$	The number of partitions.
$I.start$	The starting partition when I was added to C^k_p.
db^+	The added portion of an on-going transaction database.

3.1 IUM and FIUM Algorithms

In this section, the paper proposes two efficient algorithms, Incremental Utility Mining **(IUM)** Algorithm and Fast Incremental Utility Mining **(FIUM)** Algorithm, for incremental utility mining. **IUM**-Algorithm is based on the Two-Phase algorithm [17], [18] and **FIUM**-Algorithm is based on the ShFSM-algorithm [19]. The detail of **IUM**-Algorithm is shown in Figure 1.

In Figure 1, the **IUM**-Algorithm first finds the 1-itemset from the first partition P_1 of the original transaction database, and then it determines whether the utilities of the 1-itemsets or the transaction-weight utilities of the 1-itemsets are greater than or equal to the minimum utility threshold. If the transaction-weight utilities of the 1-itemsets are greater than or equal to the minimum utility threshold, those itemsets are saved in RU^1_1. If the utilities of the 1-itemsets are greater than or equal to the minimum utility threshold, those itemsets are saved in $LU^{[1,1]}_1$. In fact, the transaction-weight utility of an itemset is always greater than or equal to its utility. Moreover we have that $LU^{[1,1]}_1 \subseteq RU^1_1$. The 1-itemsets in RU^1_1 will be used to generate the level 2 candidates, denoted as C^1_2, which are based on Candidate-gen Algorithm. Like the previous steps, the 2-itemsets in C^1_2 with high utilities and high transaction-weight utilities will be saved in $LU^{[1,1]}_2$ and RU^1_2, respectively. The process continues until no candidate is generated in P_1. $LU^{[1,1]}$ will be saved and evaluated in the phase of P_2.

Algorithm: IUM
INPUT: $\{P_1,P_2,\ldots,P_n\}$ = database D, s = utility threshold in each partition
OUTPUT: $LU^{[1,n]}$ = High temporal utility itemsets in P_1 to P_n.

```
01  LU^[1,0]_1 = φ , for p=1,2,…,n
02  for each P_k, k=1,2,…,n
03  {
04      C^k_1 = 1-item sets in P_k
05      LU^[1,k]_1 = { I ∈ C^k_1 – LU^[1,k-1]_1 | I.u ≥ s } ∪ { I ∈ LU^[1,k-1]_1 | I.u ≥ s × (pcount – I.start + 1) }
                    // search 1-items with high utility.
06      RU^k_1 = {I ∈ C^k_1 – LU^[1,k-1]_1 | I.twu ≥ s }
                    //search 1-items with high twu.
07      for (p = 2, RU^k_p ≠ φ , p++)
08      {
09          C^k_p = Candidate-gen(RU^k_p-1) //generate candidate
10          C^k_p = C^k_p ∪ LU^[1,k-1]_p
11          LU^[1,k]_p = { I ∈ C^k_p – LU^[1,k-1]_p | I.u ≥ s } ∪ { I ∈ LU^[1,k-1]_p | I.u ≥ s × (pcount – I.start + 1) }
                    //search k-itemsets(k ≥ 2) with high utility.
12          RU^k_p = { I ∈ C^k_p – LU^[1,k-1]_p | I.twu ≥ s }
                    //search k-itemsets(k ≥ 2) with high twu.
13      }
14      LU^[1,k] = ∪ LU^k_p   //output high utility itemset in P_1 to P_n.
15  }
```

Figure 1. The procedure of IUM-Algorithm

In the temporal database $db^{[1,2]}$, which is the union of the partitions P_1 and P_2, the threshold for itemsets carried out from the previous phase (that is, the itemsets in $LU^{[1,1]}$) is threshold×2 and for newly identified itemsets is original threshold. In the same way, the algorithm will find the1-itemset from the partition P_2, and then it determines whether the utilities of the 1-itemsets or the transaction-weight utilities of the 1-itemsets are greater than or equal to the minimum utility threshold. If the transaction-weight utilities of the 1-itemsets are greater than or equal to the minimum utility threshold, those itemsets are saved in RU^2_1. If the utilities of the 1-itemsets are greater than or equal to the minimum utility threshold, those itemsets are saved in $LU^{[1,2]}_1$. Consequently, the algorithm will find the results of $LU^{[1,2]}$ and RU^2.

Repeating the process for all partitions, finally, the algorithm will output the result of $LU^{[1,n]}$ which is the set of all high temporal utility itemsets in $db^{[1,n]}$. If the value of *I.start* of itemset *I* in $LU^{[1,n]}$ is k, we also have that *I* is a high utility itemset for $db^{[k,n]}$ which is the partition database from P_k to P_n. If we want to know which itemsets are also high utility in the original transaction database ***D***, we just need to calculate the utility values of itemsets in $LU^{[1,n]}$. That is, the itemsets in $LU^{[1,n]}$ are the candidates of high utility itemsets in ***D***.

The procedure of **FIUM**-Algorithm is similar to **IUM**-Algorithm. The difference between two algorithms is the generation of candidates. **FIUM**-Algorithm also used the transaction-weight downward closure property to prune the candidates. In the step of the candidate generation for p-itemsets, **IUM**-Algorithm joins two candidates of RU^k_{p-1}, but **FIUM**-Algorithm joins the itemsets of RU^k_1 and RU^k_{p-1}. For example, consider (p-1)-itemset candidate $X_1 \in RU^k_{p-1}$, said $X_1 = \{i_1, i_2, \ldots, i_{p-1}\}$, and $X_2 \in RU^k_1$, said $X_2 = \{i_q\}$. If $i_q > i_{p-1}$, then $X = \{i_1, i_2, \ldots, i_{p-1}, i_q\}$ is a p-itemset candidate. In **IUM**-Algorithm, the time complexity of the C^k_p generation is $O(n^{2p-2})$, where n is the number of RU^k_{p-1}. However, the time complexity of **FIUM**-Algorithm generation is diminished to $O(n^p)$.

3.2 An Example for Incremental Utility Mining Algorithm

To illustrate how **IUM**-Algorithm works, we use the sample database in Table 4. In Table 4 (A), the database is first divided into three partitions $\{P_1, P_2, P_3\}$. Every partition has three transactions. The utility of each item is listed in Table 4 (B). Table 4 (C) lists the transaction-weight utility of each transaction.

Table 4. The sample database for IUM-Algorithm

(A)

		A	B	C	D	E
P_1	T_1	2	0	3	1	1
	T_2	0	1	5	0	2
	T_3	3	2	0	1	0
P_2	T_4	0	2	1	8	0
	T_5	1	0	0	2	0
	T_6	2	3	0	7	4
P_3	T_7	0	0	1	1	0
	T_8	0	2	4	5	3
	T_9	5	2	1	0	1

(B)

Item	Profit($ /unit)
A	3
B	5
C	15
D	1
E	10

(C)

TID	Transaction	TID	Transaction
T_1	62	T_7	25
T_2	100	T_8	105
T_3	34	T_9	50
T_4	33	T_{10}	16
T_5	23	T_{11}	62
T_6	68	T_{12}	23

Table 5. High temporal utility itemsets generated from database by IUM-Algorithm

$db^{[1,1]}$			
item	start	utility	twu
A	1	15	◆96
B	1	15	◆134
C	1	★120	◆162
D	1	16	◆96
E	1	30	◆162
AB	1	19	34
AC	1	★51	◆62
AD	1	31	◆96
AE	1	16	◆62
BC	1	★80	◆100
BD	1	25	34
BE	1	25	◆100
CD	1	★46	◆62
CE	1	★55	◆62
DE	1	11	◆62
ACD	1	★52	◆62
ACE	1	★61	◆62
ADE	1	17	◆62
BCE	1	★100	◆100
CDE	1	★56	◆62
ACDE	1	★62	◆62

$db^{[1,2]}$			
item	start	utility	twu
A	2	9	◆91
B	2	25	◆101
C	1	★135	◆195
D	2	35	◆124
E	2	★40	◆68
AB	2	21	◆68
AC	1	51	62
AD	2	36	◆91
AE	2	★46	◆68
BC	1	★105	◆133
BD	2	★40	◆101
BE	2	★55	◆68
CD	1	69	◆95
CE	1	55	62
DE	2	★47	◆68
ABE	2	★61	◆68
ABD	2	28	◆68
ADE	2	★53	◆68
BCD	2	33	33
BCE	1	★100	◆100
BDE	2	★62	◆68
CDE	1	56	56
ACD	1	52	62
ACE	1	61	62
ABDE	2	★68	◆68
ACDE	1	62	62

$db^{[1,3]}$			
item	start	utility	twu
A	3	15	◆50
B	3	20	◆155
C	1	★225	◆375
D	3	15	◆130
E	2	★80	◆223
AB	3	25	◆50
AC	3	30	◆50
AD	3	0	0
AE	2	71	◆118
BC	1	★200	◆288
BD	2	55	◆206
BE	2	★115	◆223
CD	3	★90	◆130
CE	3	★115	◆155
DE	2	★82	◆173
ABC	3	★40	◆50
ABE	2	★96	◆118
ACE	3	★40	◆50
ADE	2	53	68
BCD	3	★75	◆105
BCE	1	★200	◆205
BDE	2	★107	◆173
CDE	3	★95	◆105
ABCE	3	★50	◆50
BCDE	3	★105	◆105

Table 5 lists the high temporal utility itemsets and high transaction-weight utility itemsets of the temporal database. In Partition P_1, 1-itemsets {A, B, C, D, E} are found in Table 4(A). The utility value and transaction-weight utility (twu) are

calculated for each candidate itemset. We assume that minimum utility threshold is 120. Since there are three partitions, the utility threshold for P_1 is 120/3 = 40. In P_1 (or denoted as $db^{[1,1]}$), *I*.start = 1 for each itemset. Itemsets with utility or twu greater then 40 are selected. We can find $LU^{[1,1]}{}_1$={C} marked by "★", $RU^1{}_1$= {A,B,C,D,E} = $C^1{}_1$ marked by "◆". $C^1{}_2$ can be generated from the items in $RU^1{}_1$, so $C^1{}_2$= {AB, AC, AD, AE, BC, BD, BE, CD, CE, DE}. Similarly, we can find $LU^{[1,1]}{}_2$= {AC, BC, CD, CE}, $RU^1{}_2$ = {AC, AD, AE, BC, BE, CD, CE, DE}. $C^1{}_3$ can be generated from the items in $RU^1{}_2$, for example, itemset {ACD} is joined by {AC} and {AD}, and itemset {BCE} is joined by {BC} and {BE}. So $C^1{}_3$= {ACD, ACE, ADE, BCE, CDE}. By calculating utility and twu for the itemsets in $C^1{}_3$, We have $LU^{[1,1]}{}_3$= {ACD, ACE, BCE, CDE}, $RU^1{}_3$ = {ACD, ACE, BCE, CDE}. Consequently, we get $C^1{}_4$= {ACDE}, $LU^{[1,1]}{}_4$= {ACDE}, $RU^1{}_3$ = {ACDE}. In the phase of P_1, $LU^{[1,1]}$ = {C, AC, BC, CD, CE, ACD, ACE, BCE, CDE, ACDE} will be saved and evaluated in the phase of P_2.

In the temporal database $db^{[1,2]}$, which is the union of the partitions P_1 and P_2, the threshold for itemsets carried out from the previous phase is 40+40 = 80 and for newly identified itemsets is 40. For example, $LU^{[1,1]}$ = {C, AC, BC, CD, CE, ACD, ACE, BCE, CDE, ACDE} is held in $db^{[1,2]}$, and the threshold for itemsets in $LU^{[1,1]}$ is 80. Item C has C.start = 1, utility = 120+15 = 135, and twu = 162+33 = 195. Both utility and twu of C are greater than 80. Therefore, C is a high temporal utility itemset and a high transaction-weight utility itemset in $db^{[1,2]}$.

On the other hand, we find that item A has A.start = 2, utility =9, and twu = 91. Since item A does not occur in $LU^{[1,1]}$ and the twu of A is greater than the threshold 40, A is a high transaction-weight utility itemset in $db^{[1,2]}$. We can find $LU^{[1,2]}{}_1$={C, E}, $RU^2{}_1$= {A, B, C, D, E}. $C^2{}_2$ can be generated from the items in $RU^2{}_1$, so $C^2{}_2$= {AB, AC, AD, AE, BC, BD, BE, CD, CE, DE}. Consequently, we can find $LU^{[1,2]}{}_2$= {AE, BC, BD, BE, DE}, $RU^2{}_2$ = {AB, AE, BC, BD, BE, CD, DE}, $LU^{[1,2]}{}_3$= {ABE, ADE, BCE, BDE}, $RU^2{}_2$ = {ABE, ABD, ADE, BCE, BDE}, $LU^{[1,2]}{}_4$= {ABDE}, and $RU^2{}_4$ = {ABDE}. Therefore, $LU^{[1,3]}$ = {C, E, AE, BC, BD, BE, DE, ABE, ADE, BCE, BDE, ABDE}.

In the temporal database $db^{[1,3]}$, which is the union of $db^{[1,2]}$ and partition P_3, the thresholds for itemsets carried out from partition P_1 and P_2 are 40+40+40 =120 and 40+40 = 80, respectively. The threshold for newly identified itemsets is 40. In conclusion, we can discover the final result $LU^{[1,3]}$ = {C, E, BC, BE, CD, CE, DE, ABC, ABE, ACE, BCD, BCE, BDE, CDE, ABCE, BCDE}.

4. EXPERIMENTAL STUDIES

4.1 Experimental environment

All experiments were performed on a Pentium IV 3.40GHz CPU with 2 GB RAM and Microsoft Windows XP PC. **IUM**-Algorithm and **FIUM**-Algorithm were coded in Microsoft Visual C++ 6.0 and the experiment datasets were generated by IBM Data Generator [20]. The parameters of datasets were shown in Table 6.

Table 6. Parameters of datasets

Dataset	Number of transactions (ntrans)	Average transaction length (slen)	Number of items (nitems)
data.ntrans_10.nitems_0.02	10000	10	20
data.ntrans_10.nitems_0.1	10000	10	100

4.2 Performance on IUM-Algorithm and FIUM-Algorithm

In this section, the experiments first performed on the two datasets by **IUM**-Algorithm and **FIUM**-Algorithm. Tables 7 shows the generation of candidates for two algorithms in two datasets with different utility threshold $\boldsymbol{u}$, where $\boldsymbol{u}$ = α × the total utility value of Partition 1. The number of candidates of **IUM**-Algorithm and **FIUM**-Algorithm are the same in every partition. The right 3 columns of Table 7(A) and Table 7(B) list the numbers of the temporal high utility itemsets which start in each partition. Some temporal high utility itemsets generated from Partition 1 and some generated from Partition 2 and Partition 3. Because of the utilities of candidates which were generated from Partition 1 can not accumulate enough values to rise to transcend the minimum partition threshold, those candidates will be pruned in the first partition. Some new high temporal itemsets were generated from Partition 2 and Partition 3, because the utilities of those itemsets were greater than or equal to the minimum partition thresholds in Partition 2 and Partition 3. Another reason is that there are some new items which never occur in Partition 1, but the utilities of these new itemsets were greater than or equal to the minimum partition threshold.

Table 7. The number of candidate itemsets generated by IUM-Algorithm and FIUM-Algorithm

(A)

data.ntrans_10.nitems_0.02 (α = 8%)						
Candidate	IUM	FIUM	LU	Start		
	$db^{[1,1]}$			1	2	3
C_2	171	171	20	20	0	0
C_3	810	810	82	82	0	0
C_4	2033	2033	143	143	0	0
C_5	2281	2281	114	114	0	0
C_6	1385	1385	44	44	0	0
C_7	362	362	8	8	0	0
C_8	39	39	0	0	0	0
C_9	0	0	0	0	0	0
	$db^{[1,2]}$			1	2	3
C_2	190	190	23	20	3	0
C_3	936	936	99	82	17	0
C_4	3240	3240	167	143	24	0
C_5	8254	8254	133	114	19	0
C_6	13430	13430	51	44	7	0
C_7	13887	13887	9	8	1	0
C_8	9029	9029	0	0	0	0
C_9	3319	3319	0	0	0	0
C_{10}	584	584	0	0	0	0
C_{11}	33	33	0	0	0	0
C_{12}	0	0	0	0	0	0
	$db^{[1,3]}$			1	2	3
C_2	190	190	23	20	3	0
C_3	984	984	212	82	17	113
C_4	3466	3466	352	143	24	185
C_5	8599	8599	327	114	19	194
C_6	14176	14176	193	44	7	142
C_7	15831	15831	57	8	1	48
C_8	11461	11461	22	0	0	22
C_9	5199	5199	0	0	0	0
C_{10}	1281	1281	0	0	0	0
C_{11}	115	115	0	0	0	0
C_{12}	0	0	0	0	0	0

(B)

data.ntrans_10.nitems_0.1 (α = 0.6%)						
Candidate	IUM	FIUM	LU	start		
	$db^{[1,1]}$			1	2	3
C_2	4371	4371	395	395	0	0
C_3	47568	47568	235	235	0	0
C_4	29476	29476	12	12	0	0
C_5	2104	2104	0	0	0	0
C_6	51	51	0	0	0	0
C_7	0	0	0	0	0	0
	$db^{[1,2]}$			1	2	3
C_2	4753	4753	445	395	50	0
C_3	88926	88926	300	235	65	0
C_4	234453	234453	19	12	7	0
C_5	68654	68654	0	0	0	0
C_6	4020	4020	0	0	0	0
C_7	214	214	0	0	0	0
C_8	31	31	0	0	0	0
C_9	3	3	0	0	0	0
C_{10}	0	0	0	0	0	0
	$db^{[1,3]}$			1	2	3
C_2	4851	4851	608	395	50	163
C_3	108669	108669	602	235	65	302
C_4	571774	571774	129	12	7	110
C_5	387832	387832	0	0	0	0
C_6	49057	49057	0	0	0	0
C_7	2345	2345	0	0	0	0
C_8	235	235	0	0	0	0
C_9	30	30	0	0	0	0
C_{10}	2	2	0	0	0	0
C_{11}	0	0	0	0	0	0

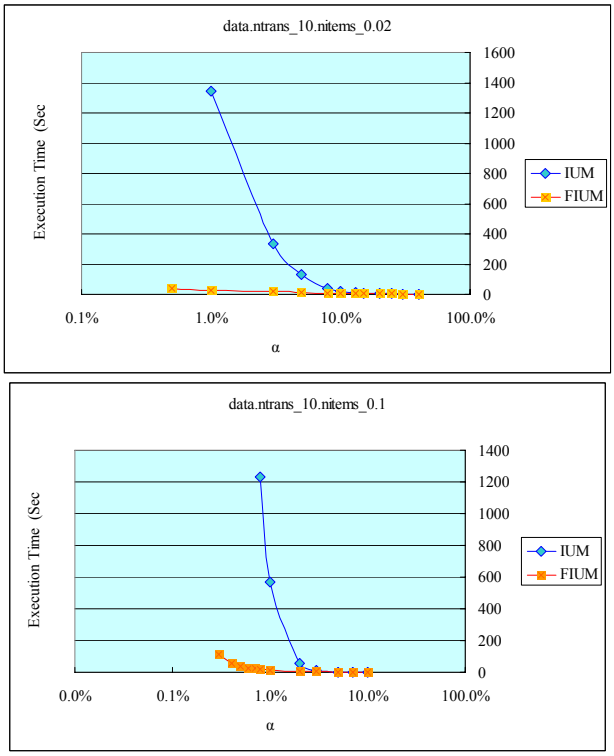

Figure 2. Execution time for IUM-Algorithm and FIUM-Algorithm with various minimum utility thresholds in two datasets

Figure 2 shows that the performance in the execution times of **IUM**-Algorithm and **FIUM**-Algorithm on two datasets with various minimum utility thresholds. The **FIUM**-Algorithm is more efficiently than **IUM**-Algorithm by the experimental result. Because of **FIUM**-Algorithm spends less time in joining candidate itemsets than **IUM**-Algorithm does.

5. CONCLUSIONS

The study proposed novel incremental utility mining methods, IUM-Algorithm, and FIUM-Algorithm, which can discover temporal high utility itemsets. The algorithms can efficiently identify all high temporal utility itemsets that users will be interested in particular periods when the new transaction data are added into the original transaction database. The algorithm not only can find the temporal high utility itemsets for particular time periods, but also can find the high utility itemsets for the entire transaction database. In fact, the high temporal utility itemsets are the candidates of high utility itemsets.

Incremental utility mining strives for finding high temporal utility itemsets that drive large portions of utility in a particular period. However, it does not indicate how often such itemsets appear in the period. In the future, the temporal utility-frequent mining can be more investigated.

6. REFERENCES

[1] Agrawal, R. and Srikant, R. 1994. Fast Algorithms for Mining Association Rules. In Proceedings of the 20th VLDB Conference (Santiago, Chile). 487-499.

[2] Park, J. S., Chen, M. S., and Yu, P. S. 1995. An Effective Hash-based Algorithm for Mining Association Rules. In Proceedings of ACM SIGMOD Conf. on Management of Data. 175-186.

[3] Savasere, A., Oiecinski, E., and Navathe, S. 1995. An Efficient Algorithm for Mining Association Rules in Large Databases. In Proceedings of the 20th Int. Conf. on Very Large Data Bases. 432-444.

[4] Agrawal, R. and Srikant, R. 1996. Parallel Mining of Association Rules. IEEE Transactions on Knowledge and Data Engineering. 8, 6, 962-969.

[5] Agrawal, R. and Shim, K. 1996. Developing Tightly-Coupled Data Mining Applications on a Relational Database System. In Proceedings of the 2nd Int. Conf. on Knowledge Discovery in Databases and Data Mining (Portland, Oregon). 287-290.

[6] Agrawal, R., Srikant, R., and Vu, Q. 1997. Mining Association Rules with Item Constraints. In Proceedings of the 3rd Int. Conf. on Knowledge Discovery in Database and Data Mining (Newport Beach, California). 67-73.

[7] Cheung, D., Han, J., Ng V., and Wong, C. Y. 1996. Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique. In Proceedings of the 12th Int. Conf. on Data Engineering (ICDE96). 106-114.

[8] Cheung, D., Lee, S.D., and Kao, B. 1997. A General Incremental Technique for Updating Discovered Association Rules. In Proceedings of Int. Conf. on Database Systems for Advanced Applications. 185-194.

[9] Hong, T. P., Wang, C. Y., and Tao, Y. H. 2000. Incremental Data Mining Based on Two Support Thresholds. In Proceedings of the 4th Int. Conf. on Knowledge-Based Intelligent Engineering Systems and Allied Technologies. 436-439.

[10] Lee, C. H., Lin, C. R., and Chen, M. S. 2001. Sliding-Window Filtering: An Efficient Algorithm for Incremental Mining. In Proceedings of the 10th Int. Conf. on Information and Knowledge Management. 263-270.

[11] Chang, C. H., Lin, C. R., and Chen, M. S. 2001. On Mining General Temporal Association Rules in A Publication Database. In Proceedings of 2001 IEEE Int. Conf. on Data Mining. 337-344.

[12] Masseglia, F., Poncelet, P., and Teisseire, M. 2003. Incremental Mining of Sequential Patterns in Large Databases. Data & Knowledge Engineering. 46. 97-121.

[13] Cheng, H., Yang, X., and Han, J. 2004. IncSpan: Incremental Mining of Sequential Patterns in Large Database. In Proceedings of the 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining. 527-532.

[14] Yao, H., Hamilton, H. J., and Butz, C. J. 2004. A Foundational Approach to Mining Itemset Utilities from Databases. In Proceedings of the 4th SIAM Int. Conf. on Data Mining. 428-486.

[15] Tseng, V. S., Chu, C. J., and Liang, T. 2006. Efficient Mining of Temporal High Utility Itemsets from Data

Streams. In Proceedings of ACM KDD Workshop on Utility-Based Data Mining.

[16] Liu, Y., Liao W. K., and Choudhary, A. 2005. A Fast High Utility Itemsets Mining Algorithm. In Proceedings of the 11th ACM SIGKDD Workshop on Utility-Based Data Mining.

[17] Liu, Y., Liao, W. K., and Choudhary, A. 2005. A Two-Phase Algorithm for Fast Discovery of High Utility Itemsets. LECTURE NOTES IN COMPUTER SCIENCE 3518, Springer-Verlag, Berlin Heidelberg. 689-695.

[18] Li, Y. C., Yeh, J. S., and Chang, C. C. 2005. A Fast Algorithm for Mining Share-Frequent Itemsets. In Proceedings of the 7th Asia Pacific Web Conference, Springer-Verlag, Germany. 417-428.

[19] Li, Y. C., Yeh, J. S., and Chang, C. C. 2005. Efficient Algorithm for Mining Share-Frequent Itemsets. In Proceedings of Fuzzy Logic, Soft Computing and Computational Intelligence (IFSA 2005), Springer Tsinghua, China. 534-539.

[20] IBM Almaden Research Center, http://www.almaden.ibm.com/cs/projects/iis/hdb/Projects/data_mining/mining.shtml