Efficient Mining of Temporal High Utility Itemsets from Data streams

Vincent S. Tseng
Dept. Computer Science and Information Engineering
National Cheng Kung University, Taiwan, ROC
tsengsm@mail.ncku.edu.tw

Chun-Jung Chu
Dept. of Computer Science
National Chiao Tung University, Taiwan, ROC
cjchu@cis.nctu.edu.tw

Tyne Liang
Dept. of Computer Science
National Chiao Tung University, Taiwan, ROC
tliang@cis.nctu.edu.tw

ABSTRACT
Utility itemsets are considered as the different values of individual items as utilities, and utility mining aims at identifying the itemsets with high utilities. The temporal high utility itemsets are the itemsets with support larger than a pre-specified threshold in current time window of data stream. Discovery of temporal high utility itemsets is an important process for mining interesting patterns like association rules from data streams. In this paper, we propose a novel method, namely *THUI* (Temporal High Utility Itemsets) *-Mine*, for mining temporal high utility itemsets from data streams efficiently and effectively. To our best knowledge, this is the first work on mining temporal high utility itemsets from data streams. The novel contribution of *THUI-Mine* is that it can effectively identify the temporal high utility itemsets by generating fewer temporal high transaction-weighted utilization 2-itemsets such that the execution time can be reduced substantially in mining all high utility itemsets in data streams. In this way, the process of discovering all temporal high utility itemsets under all time windows of data streams can be achieved effectively with limited memory space, less candidate itemsets and CPU I/O time. This meets the critical requirements on time and space efficiency for mining data streams. The experimental results show that *THUI-Mine* can discover the temporal high utility itemsets with higher performance and less candidate itemsets compared to other algorithms under various experimental conditions.

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

General Terms
Algorithms, Design

Keywords
utility mining, temporal high utility itemsets, data streams, association rules

UBDM'06, August 20, 2006, Philadelphia, Pennsylvania, USA.

1. INTRODUCTION
The mining of association rules for finding the relationship between data items in large databases is a well studied technique in data mining field with representative methods like *Apriori* [1, 2]. The problem of mining association rules can be decomposed into two steps. The first step involves finding all frequent itemsets (or say large itemsets) in databases. Once the frequent itemsets are found, generating association rules is straightforward and can be accomplished in linear time.

An important research issue extended from the association rules mining is the discovery of temporal association patterns in data streams due to the wide applications on various domains. Temporal data mining can be defined as the activity of looking for interesting correlations or patterns in large sets of temporal data accumulated for other purposes [6]. For a database with a specified transaction window size, we may use the algorithm like Apriori to obtain frequent itemsets from the database. For time-variant data streams, there is a strong demand to develop an efficient and effective method to mine various temporal patterns [11]. However, most methods designed for the traditional databases cannot be directly applied for mining temporal patterns in data streams because of the high complexity.

In many applications, we would like to mine temporal association patterns in data streams for amount of most recent data. That is, in the temporal data mining, one has to not only include new data (i.e., data in the new hour) into, but also remove the old data (i.e., data in the most obsolete hour) from the mining process. Without loss of generality, consider a typical market-

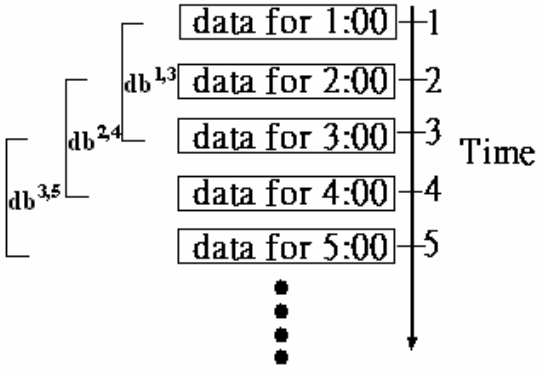


Figure 1. An example of online transaction flows.

basket application as illustrated in Figure 1 has been considered. The transaction flow in such an application is shown in Figure 1 where transaction data purchased by customers as time advances.

In Figure 1, for example, data was accumulated with time passing by. Old data in the past hours becomes useless for reference. People might be most interested in the temporal association patterns in the latest three hours (i.e., $db^{3,5}$) as shown in Figure 1. It can be seen that in such a data stream environment it is intrinsically difficult to conduct the frequent pattern identification due to the constraints of limited time and space. Furthermore, it takes considerable time to find temporal frequent itemsets in different time windows. However, the frequency of an itemset may not be a sufficient indicator of interestingness, because it only reflects the number of transactions in the database that contain the itemset. It does not reveal the *utility* of an itemset, which can be measured in terms of cost, profit, or other expressions of user preference. On the other hand, frequent itemsets may only contribute a small portion of the overall profit, whereas non-frequent itemsets may contribute a large portion of the profit. In reality, a retail business may be interested in identifying its most valuable customers (customers who contribute a major fraction of the profits to the company). Hence, frequency is not sufficient to answer questions, such as whether an itemset is highly profitable, or whether an itemset has a strong impact. Utility mining is thus useful in a wide range of practical applications and was recently studied in [7, 14, 19]. This also motivates our research in developing a new scheme for finding *temporal high utility itemsets (THUI)* from data streams.

Table 1. A transaction database and its utility table.

(a) Transaction table

			TID \ ITEM	A	B	C	D	E	
$db^{1,3}$	$\triangle^-$	P_1	T_1	0	0	26	0	1	
			T_2	0	6	0	1	1	
			T_3	12	0	0	1	0	
	D^-	P_2	T_4	0	1	0	7	0	
			T_5	0	0	12	0	2	
			T_6	1	4	0	0	1	
		P_3	T_7	0	10	0	0	1	
			T_8	1	0	1	3	1	$db^{2,4}$
			T_9	1	1	27	0	0	
	$\triangle^+$	P_4	T_{10}	0	6	2	0	0	
			T_{11}	0	3	0	2	0	
			T_{12}	0	2	1	0	0	

(b) The utility table

ITEM	PROFIT($)(per unit)
A	3
B	10
C	1
D	6
E	5

Recently, a *utility mining* model was defined in [19]. Utility is a measure of how "useful" (i. e. "profitable") an itemset is. The definition of utility of an itemset X, $u(X)$, is the sum of the utilities of X in all the transactions containing X. The goal of utility mining is to identify high utility itemsets which drive a large portion of the total utility. Traditional association rules mining model assumes that the utility of each item is always 1 and the sales quantity is either 0 or 1, thus it is only a special case of utility mining, where the utility or the sales quantity of each item could be any number. If $u(X)$ is greater than a utility threshold, X is a high utility itemset. Otherwise, it is a low utility itemset. Table 1 is an example of utility mining in a transaction database. The number in each transaction in Table 1(a) is the sales volume of each item, and the utility of each item is listed in Table 1(b). For example, $u(\{B, D\}) = (6\times10+1\times6) + (1\times10+7\times6) + (3\times10+2\times6) = 160$. {B, D} is a high utility itemset if the utility threshold is set at 120.

However, a high utility itemset may consist of some low utility items. Another attempt is to adopt the level-wise searching schema that exists in fast algorithms, such as Apriori [3]. However, this algorithm doesn't apply to the *utility mining* model. For example, $u(D) = 84 < 120$, D is a low utility item, but its superset {B, D} is a high utility itemset. If using Apriori to find high utility itemset, all the combinations of all the items must be generated. Moreover, to discover a long pattern, the number of candidates is exorbitantly large. The cost of either computation time or memory is intolerable, regardless of what implementation is applied. The challenge of utility mining is not only in restricting the size of the candidate set but simplifying the computation for calculating the utility. Another challenge of utility mining is how to find temporal high utility itemsets from data streams as time advances.

In this paper, we explore the issue of efficiently mining high utility itemsets in temporal databases like data streams. We propose an algorithm named *THUI-Mine* that can discover temporal high utility itemsets from data streams efficiently and effectively. The underlying idea of *THUI-Mine* algorithm is to integrate the advantages of Two-Phase algorithm [14] and SWF algorithm [12] and augment with the incremental mining techniques for mining temporal high utility itemsets efficiently. The novel contribution of *THUI-Mine* is that it can efficiently identify the utility itemsets in data streams so that the execution time for mining high utility itemsets can be substantially reduced. That is, *THUI-Mine* can discover the temporal high utility itemsets in current time window and also discover the temporal high utility itemsets in the next time window with limited memory space and less computation time by sliding window filter method. In this way, the process of discovering all temporal high utility itemsets under all time windows of data streams can be

achieved effectively under limited memory space, less candidate itemsets and CPU I/O. This meets the critical requirements of time and space efficiency for mining data streams. Through experimental evaluation, *THUI-Mine* is shown to produce fewer candidate itemsets in finding the temporal high utility itemsets, so it outperforms other methods in terms of execution efficiency. Moreover, it also achieves high scalability in dealing with large databases. To our best knowledge, this is the first work on mining temporal high utility itemsets from data streams.

The rest of this paper is organized as follows: Section 2 overviews the related work. Section 3 describes the proposed approach, *THUI-Mine*, for finding the temporal high utility itemsets. In section 4, we describe the experimental results for evaluating the proposed method. The conclusion of the paper is provided in Section 5.

2. RELATED WORK

In association rules mining, Apriori [3], DHP [15], and partition-based ones [13, 16] were proposed to find frequent itemsets. Many important applications have called for the need of incremental mining. This is due to the increasing use of the record-based databases whose data are being continuously added. Many algorithms like FUP [8], FUP_2 [9] and UWEP [4, 5] are proposed to solve incremental database for finding frequent itemsets. The FUP algorithm updates the association rules in a database when new transactions are added to the database. Algorithm FUP is based on the framework of Apriori and is designed to discover the new frequent itemsets iteratively. The idea is to store the counts of all the frequent itemsets found in a previous mining operation. Using these stored counts and examining the newly added transactions, the overall count of these candidate itemsets are then obtained by scanning the original database. An extension to the work in [8] was reported in [9] where the authors propose an algorithm FUP_2 for updating the existing association rules when transactions are added to and deleted from the database. UWEP (Update With Early Pruning) is an efficient incremental algorithm, that counts the original database at most once, and the increment exactly once. In addition the number of candidates generated and counted is minimum.

In recent years, processing data from data streams is a very popular topic in data mining. Many algorithms like FTP-DS [17] and RAM-DS [18] are proposed to process data in data streams. FTP-DS is a regression-based algorithm to mine frequent temporal patterns for data streams. A wavelet-based algorithm, called algorithm RAM-DS, to perform pattern mining tasks for data streams by exploring both temporal and support count granularities.

Some algorithms like SWF [12] and Moment [10] were proposed to find frequent itemsets over a stream sliding window. By partitioning a transaction database into several partitions, algorithm SWF employs a filtering threshold in each partition to deal with the candidate itemset generation. Moment algorithm use the closed enumeration tree (CET), to maintain a dynamically selected set of itemsets over a sliding window.

A formal definition of utility mining and theoretical model was proposed in [19], namely MEU, where the utility is defined as the combination of utility information in each transaction and additional resources. Since this model cannot rely on downward closure property of Apriori to restrict the number of itemsets to be examined, a heuristics is used to predict whether an itemset should be added to the candidate set. However, the prediction usually overestimates, especially at the beginning stages, where the number of candidates approaches the number of all the combinations of items. The examination of all the combinations is impractical, either in computation cost or in memory space cost, whenever the number of items is large or the utility threshold is low. Although this algorithm is not efficient or scalable, it is by far the best to solve this specific problem.

Another algorithm named Two-Phase was proposed in [14], which is based on the definition in [19] and achieves for finding high utility itemsets. It presented a Two-Phase algorithm to prune down the number of candidates and can obtain the complete set of high utility itemsets. In the first phase, a model that applies the "transaction-weighted downward closure property" on the search space to expedite the identification of candidates. In the second phase, one extra database scan is performed to identify the high utility itemsets. However, this algorithm must rescan the whole database when added new transactions from data streams. It need more times on processing I/O and CPU cost for finding high utility itemsets. Hence, Two-Phase algorithm is just only focused on traditional databases and is not suited for data streams.

Although there existed numerous studies on high utility itemsets mining and data stream analysis as described above, there is no algorithm proposed for finding temporal high utility itemsets in data streams. This motivates our exploration on the issue of efficiently mining high utility itemsets in temporal databases like data streams in this research.

3. PROPOSED METHOD: THUI-MINE

The goal of utility mining is to discover all the itemsets whose utility values are beyond a user specified threshold in a transaction database. Utility mining is to find all the high utility itemsets in [19]. An itemset X is a *high utility itemset* if $u(X) \geq \varepsilon$, where $X \subseteq I$ and ε is the minimum utility threshold, otherwise, it is a *low utility itemset*. For example, in Table 1, $u(A, T_8) = 1\times3 = 3$, $u(\{A, C\}, T_8) = u(A, T_8) + u(C, T_8) = 1\times3 + 1\times1 = 4$, and $u(\{A, C\}) = u(\{A, C\}, T_8) + u(\{A, C\}, T_9) = 4 + 30 = 34$. If $\varepsilon = 120$, {A, C} is a low utility itemset. However, if an item is a low utility item, its superset may be a high utility itemset. For example, $u(D) = 84 < 120$, D is a low utility item, but its superset {B, D} is a high utility itemset because of $u(\{B, D\}) = 160 > 120$. Hence, all the combinations of all items should be processed so that it never loses any high utility itemset. But the cost of either computation time or memory is intolerable.

Liu *et al* [14] proposed Two-Phase algorithm for pruning candidate itemsets and simplify the calculation of utility. First, Phase I overestimates some low utility itemsets, but it never underestimates any itemsets. For the example in Table 1, the *transaction utility of transaction Tq*, denoted as $tu(Tq)$, is the sum of the utilities of all items in *Tq* and 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, then $twu(A) = tu(T_3) + tu(T_6) + tu(T_8) + tu(T_9) = 42 + 48 + 27 + 40 = 157$ and $twu(\{D, E\}) = tu(T_2) + tu(T_8) = 71 + 27 = 98$. In fact, $u(A) = u(\{A\}, T_3) + u(\{A\}, T_6) + u(\{A\}, T_8) + u(\{A\}, T_9) = 36 + 3 + 3 + 3 = 45$ and $u(\{D, E\}) = u(\{D, E\}, T_2) + u(\{D, E\}, T_8) = 11 + 23 = 34$. So Phase I overestimates some low utility itemsets, it never

underestimates any itemsets. Table 2 gives the transaction utility for each transaction in Table 1. Second, one extra database scan is performed to filter the overestimated itemsets in phase II. For example, $twu(A) = 157 > 120$ but $u(A) = 45 < 120$. Then item {A} is pruned. Otherwise, it is a high utility itemset. Finally, all of high utility itemsets are discovered by this way.

Table 2. Transaction utility of the transaction database.

TID	Transaction Utility	TID	Transaction Utility
T_1	31	T_7	105
T_2	71	T_8	27
T_3	42	T_9	40
T_4	52	T_{10}	62
T_5	22	T_{11}	42
T_6	48	T_{12}	21

Our algorithm *THUI-Mine* is based on the principle of Two-Phase algorithm [14], and we extend it with the sliding-window-filtering technique and focus on utilizing incremental methods to improve the response time with fewer candidate itemsets and CPU I/O. In essence, by partitioning a transaction database into several partitions from data streams, algorithm *THUI-Mine* employs a filtering threshold in each partition to deal with the transaction-weighted utilization itemsets generation. The cumulative information in the prior phases is selectively carried over toward the generation of transaction-weighted utilization itemsets in the subsequent phases by *THUI-Mine*. In the processing of a partition, a progressive transaction-weighted utilization set of itemsets is generated by *THUI-Mine*. Explicitly, a progressive transaction-weighted utilization set of itemsets is composed of the following two types of transaction-weighted utilization itemsets, i.e., (1) the transaction-weighted utilization itemsets that were carried over from the previous progressive candidate set in the previous phase and remain as transaction-weighted utilization itemsets after the current partition is taken into consideration and (2) the transaction-weighted utilization itemsets that were not in the progressive candidate set in the previous phase but are newly selected after only taking the current data partition into account As such, after the processing of a phase, algorithm *THUI-Mine* outputs a cumulative filter, denoted by CF, which consists of a progressive transaction-weighted utilization set of itemsets, their occurrence counts and the corresponding partial support required. With these design considerations, algorithm *THUI-Mine* is shown to have very good performance for mining temporal high utility itemsets from data streams. In Section 3.1, we give an example for mining temporal high utility itemsets from data stream. The proposed algorithm, *THUI-Mine*, is described in details in Section 3.2.

3.1 An Example for Mining Temporal High Utility Itemsets

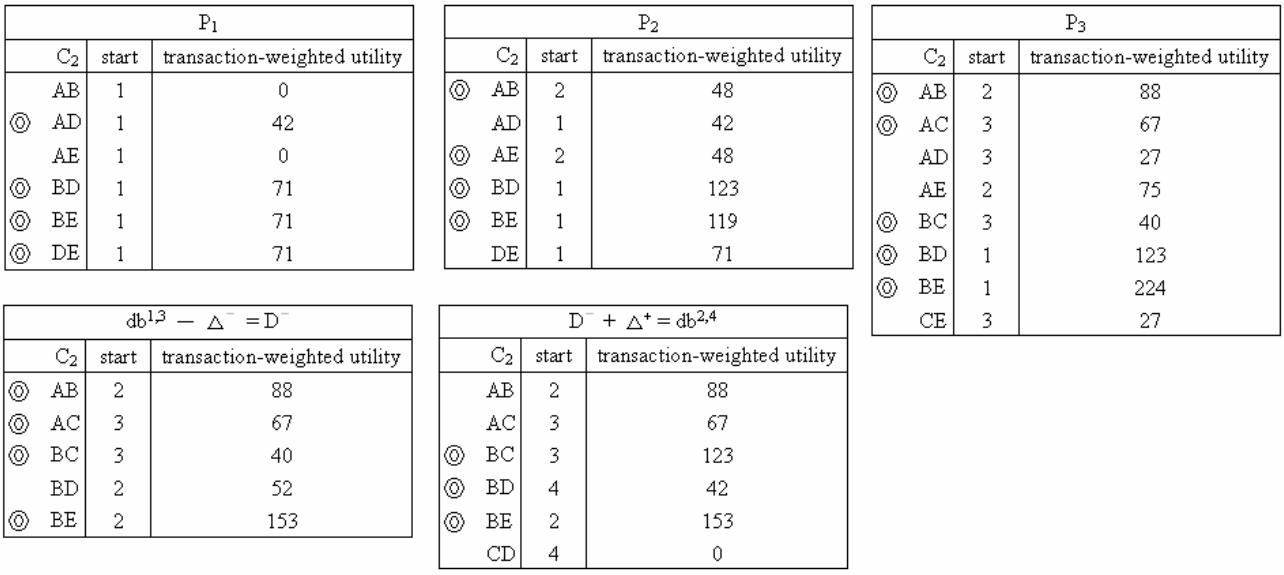
The proposed *THUI-Mine* algorithm can be best understood by the illustrative transaction database in Table 1 and Figure 2 where a scenario of generating high utility itemsets from data streams for mining temporal high utility itemsets is given. We set the utility threshold as 120 with nine transactions. Without loss of generality, the temporal mining problem can be decomposed into two procedures:

1. Preprocessing procedure: This procedure deals with mining on the original transaction database.

2. Incremental procedure: The procedure deals with the update of the high utility itemsets form data streams.

The preprocessing procedure is only utilized for the initial utility mining in the original database, e.g., $db^{1,n}$. For the generation of mining high utility itemsets in $db^{2,n+1}$, $db^{3,n+2}$, $db^{i,j}$, and so on, the incremental procedure is employed. Consider the database in Table 1. Assume that the original transaction database $db^{1,3}$ is segmented into three partitions, i.e., $\{P_1, P_2, P_3\}$, in the pre processing procedure. Each partition is scanned sequentially for the generation of candidate 2-itemsets in the first scan of the database $db^{1,3}$. After scanning the first segment of 3 transactions, i.e., partition P_1, 2-itemsets {AB, AD AE, BD, BE, DE} are generated as shown in Figure 2. In addition, each potential candidate itemset $c \in C_2$ has two attributes: (1) c.start which contains the identity of the starting partition when c was added to C_2, and (2) transaction-weighted utility which is the sum of the transaction utilities of all the transactions containing c since c was added to C_2. Since there are three partitions, the utility threshold of each partition is $120 / 3 = 40$. Such a partial utility threshold is called the filtering threshold in this paper. Itemsets whose transaction-weighted utility are below the filtering threshold are removed. Then, as shown in Figure 2, only {AD, BD, BE, DE}, marked by "◎", remain as temporal high transaction-weighted utilization 2-itemsets whose information is then carried over to the next phase of processing. Similarly, after scanning partition P_2, the temporal high transaction-weighted utilization 2-itemsets are recorded.

From Figure 2, it is noted that since there are also 3 transactions in P_2, the filtering threshold of those itemsets carried out from the previous phase is $40 + 40 = 80$ and that of newly identified candidate itemsets is 40. It can be seen from Figure 2 that we have 4 temporal high transaction-weighted utilization 2-itemsets in C_2 after the processing of partition P_2, and 2 of them are carried from P_1 to P_2 and 2 of them are newly identified in P_2. Finally, partition P_3 is processed by algorithm *THUI-Mine*. The resulting temporal high transaction-weighted utilization 2-itemsets are {AB, AC, BC, BD, BE} as shown in Figure 2. Note that though appearing in the previous phase P_2, itemset {AE} is removed from temporal high transaction-weighted utilization 2-itemsets once P_3 is taken into account since its transaction-weighted utility does not meet the filtering threshold then, i.e., $75 < 120$. However, we do have two new itemset, i.e., AC and BC, which join the C_2 as temporal high transaction-weighted utilization 2-itemsets. Consequently, we have 5 temporal high transaction-weighted utilization 2-itemsets generated by *THUI-Mine*, and 2 of them are carried from P_1 to P_3, 1 of them is carried from P_2 to P_3 and 2 of them are newly identified in P_3. Note that instead of 10 candidate itemsets that would be generated if Two-Phase algorithm were used, only 5 temporal high transaction-weighted utilization 2-itemsets are generated by *THUI-Mine*. After processing P_1 to P_3, those temporal high transaction-



P_1			
	C_2	start	transaction-weighted utility
	AB	1	0
◎	AD	1	42
	AE	1	0
◎	BD	1	71
◎	BE	1	71
◎	DE	1	71

P_2			
	C_2	start	transaction-weighted utility
◎	AB	2	48
	AD	1	42
◎	AE	2	48
◎	BD	1	123
◎	BE	1	119
	DE	1	71

P_3			
	C_2	start	transaction-weighted utility
◎	AB	2	88
◎	AC	3	67
	AD	3	27
	AE	2	75
◎	BC	3	40
◎	BD	1	123
◎	BE	1	224
	CE	3	27

$db^{1,3} - \Delta^- = D^-$			
	C_2	start	transaction-weighted utility
◎	AB	2	88
◎	AC	3	67
◎	BC	3	40
	BD	2	52
◎	BE	2	153

$D^- + \Delta^+ = db^{2,4}$			
	C_2	start	transaction-weighted utility
	AB	2	88
	AC	3	67
◎	BC	3	123
◎	BD	4	42
◎	BE	2	153
	CD	4	0

Figure 2. Temporal high utility itemsets generated from data streams by THUI-Mine.

weighted utilization itemsets in $db^{1,3}$ are {A, B, C, D, E, AB, AC, BC, BD, BE}.

After generating temporal high transaction-weighted utilization 2-itemsets from the first scan of database $db^{1,3}$, we employ the scan reduction technique and use temporal high transaction-weighted utilization 2-itemsets to generate C_k (k = 3, 4, ..., n), where C_n is the candidate last itemsets. It can be verified that temporal high transaction-weighted utilization 2-itemsets generated by *THUI-Mine* can be used to generate the candidate 3-itemsets. Clearly, a C_3 generated from temporal high transaction-weighted utilization 2-itemsets. For example, 3-candidate itemset {ABC} is generated from temporal high transaction-weighted utilization 2-itemsets {AB, AC, BC} in $db^{1,3}$. However, the temporal high transaction-weighted utilization 2-itemsets generated by *THUI-Mine* is very close to the high utility itemsets. Similarly, all C_k can be stored in main memory, and we can find temporal high utility itemsets together when the second scan of the database $db^{1,3}$ is performed. Thus, only two scans of the original database $db^{1,3}$ are required in the preprocessing step. The resulting temporal high utility itemsets are {B} and {BE} because u(B) = 330 >120 and u({B, E}) = 215 > 120 .

One important merit of *THUI-Mine* mainly lies in its incremental procedure. As depicted in Figure 2, the mining database will be moved from $db^{1,3}$ to $db^{2,4}$. Thus, some transactions, i.e., T_1, T_2, and T_3, are deleted from the mining database and other transactions, i.e., T_{10}, T_{11}, and T_{12}, are added. To illustrate more clearly, this incremental step can also be divided into three sub-steps: (1) generating temporal high transaction-weighted utilization 2-itemsets in $D^- = db^{1,3} - \Delta^-$, (2) generating temporal high transaction-weighted utilization 2-itemsets in $db^{2,4} = D^- + \Delta^+$ and (3) scanning the database $db^{2,4}$ only once for the generation of all temporal high utility itemsets. In the first sub-step, $db^{1,3} - \Delta^- = D^-$, we check out the pruned partition P_1, and reduce the value of transaction-weighted utility and set c.start = 2 for those temporal transaction-weighted utilization 2-itemsets where c.start = 1. It can be seen that itemset {BD} were removed. Next, in the second sub-step, we scan the incremental transactions in P_4. The process in $D^- + \Delta^+ = db^{2,4}$ is similar to the operation of scanning partitions, e.g., P_2, in the preprocessing step. The new itemset {BD} join the temporal high transaction-weighted utilization 2-itemsets after the scan of P_4. In the third sub-step, we use temporal high transaction-weighted utilization 2-itemsets to generate C_k as mentioned above. Finally, those temporal high transaction-weighted utilization itemsets in $db^{2,4}$ are {B, C, D, E, BC, BD, BE}. With scanning $db_{2,4}$ only once, *THUI-Mine* obtains temporal high utility itemsets {B, BC, BE} in $db^{2,4}$.

Table 3. Meanings of symbols used.

$db^{i,j}$	Partition_database (D) from P_i to P_j
s	Utility threshold in one partition
$\| P_k \|$	Number of transactions in partition P_k
TUP_k (I)	Trans. in P_k that contain itemset I with transaction utility
UP_k (I)	Trans. in P_k that contain itemset I with utility
$\| db^{1,n},(I) \|$	Trans. No. in $db^{1,n}$ that contain itemset I
$C^{i,j}$	The progressive candidate sets of $db^{i,j}$
$Thtw^{i,j}$	The progressive temporal high transaction-weighted utilization 2-itemsets of $db^{i,j}$
$Thu^{i,j}$	The progressive temporal high utility itemsets of $db^{i,j}$
Δ^-	The deleted portion of an ongoing database
D^-	The unchanged portion of an ongoing database
Δ^+	The added portion of an ongoing database

3.2 THUI-Mine Algorithm

For easier illustration, the meanings of various symbols used are given in Table 3. The preprocessing procedure and the incremental procedure of algorithm *THUI-Mine* are described in Section 3.2.1 and Section 3.2.2, respectively.

3.2.1 Preprocessing procedure of THUI-Mine

The preprocessing procedure of Algorithm *THUI-Mine* is shown in Figure 3. Initially, the database $db^{1,n}$ is partitioned into n partitions by executing the preprocessing procedure (in Step 2), and CF, i.e., cumulative filter, is empty (in Step 3). Let $Thtw^{1,n}$ be the set of progressive temporal high transaction-weighted utilization 2-itemsets of $db^{i,j}$. Algorithm *THUI-Mine* only records $Thtw^{1,n}$ which is generated by the preprocessing procedure to be used by the incremental procedure. From Step 4 to Step 16, the algorithm processes one partition at a time for all partitions. When partition P_i is processed, each potential candidate 2-itemset is read and saved to CF. The transaction-weight utility of an itemset I and its starting partition are recorded in I.twu and I.start, respectively. An itemset, whose I.twu ≥ s, will be kept in CF. Next, we select $Thtw^{1,n}$ from I where I∈ CF and keep I.twu in main memory for the subsequent incremental procedure. With employing the scan reduction technique from Step 19 to Step 26, $C_h^{1,n}$ (h ≥ 3) are generated in main memory. After refreshing I.count = 0 where I.twu = 0 where I∈ $Thtw^{1,n}$, we begin the last scan of database for the preprocessing procedure from Step 28 to Step 31. Finally, those itemsets satisfying the constraint that I.u ≧ s×P.count are finally obtained as the temporal high utility itemsets.

```
1. n = Number of partitions;
2. |db^{1,n}| = Σ_{k=1,n} |P_k|;
3. CF = φ;
4. begin for k = 1 to n                    // 1st scan of db^{1,n}
5.       begin for each 2-itemset I ∈ P_k
6.             if ( I ∉ CF )
7.                 I.twu = TUP_k (I);
8.                 I.start = k;
9.                 if (I.twu ≥ s×P.count )     // P.count is number of partitions
10.                      CF = CF ∪ I;
11.            if ( I ∈ CF )
12.                I.twu = I.twu + TUP_k (I);
13.                if (I.twu < s×P.count )
14.                     CF = CF − I;
15.      end
16. end
17. select Thtw^{1,n} from I where I ∈ CF;
18. keep Thtw^{1,n} in main memory;
19. h = 2;
20. begin while (Thtw^{1,n} ≠ φ)       //Database scan reduction
21.    if (h=2)
22.       C_{h+1}^{1,n} = Thtw^{1,n} * Thtw^{1,n} ;
23.    else
24.       C_{h+1}^{1,n} = C_h^{1,n} * C_h^{1,n}
25.    h = h + 1;
26. end
27. refresh I.twu = 0 where I ∈ Thtw^{1,n} ;
28. begin for k = 1 to n                   //2nd scan of db^{1,n}
29.      for each itemset I ∈ Thtw^{1,n} ∪ C_{h+1}^{1,n}
30.           I.u = I.u + UP_k (I);
31. end
32. for each itemset I ∈ Thtw^{1,n} ∪ C_{h+1}^{1,n}
33.      if (I.u ≧ s×P.count )
34.           Thu^{1,n} = Thu^{1,n} ∪ I;
35. end
36. return Thu^{1,n};
```

Figure 3. Preprocessing procedure of THUI-Mine.

```
1. Original database = db^{m,n};
2. New database = db^{i,j} ;
3. Database removed Δ^− = Σ_{k=m,i-1} P_k;
4. Database added Δ^+ = Σ_{k=n+1,j} P_k;
5. D^− = Σ_{k=i,n} P_k;
6. db^{i,j} = db^{m,n} −Δ^− + Δ^+;
7. loading Thtw^{m,n} of db^{m,n} into CF where I ∈ Thtw^{m,n};
8. begin for k = m to i − 1 // one scan of Δ^−
9.      begin for each 2-itemset I ∈ P_k
10.          if ( I ∈ CF and I.start ≤ k )
11.             I.twu = I.twu  −  TUP_k (I);
12.             I.start = k + 1;
13.             if ( I.twu < s×P.count )
14.             CF = CF − I;
15.     end
16. end
17. begin for k = n +1 to j // one scan of Δ^+
18.     begin for each 2-itemset I ∈ P_k
19.          if ( I ∉ CF )
20.             I.twu = TUP_k (I);
21.             I.start = k;
22.             if (I.twu ≥ s×P.count )
23.                 CF = CF ∪ I;
24.             if ( I ∈ CF )
25.                 I.twu = I.twu + TUP_k (I);
26.                 if (I.twu < s×P.count)
27.                     CF = CF − I;
28.     end
29. end
30. select Thtw^{i,j} from I where I ∈ CF;
31. keep Thtw^{i,j} in main memory;
32. h = 2
33. begin while (Thtw^{i,j} ≠ φ)       //Database scan reduction
34.     if (h=2)
35.        C_{h+1}^{i,j} = Thtw^{i,j} * Thtw^{i,j} ;
36.     else
37.        C_{h+1}^{i,j} = C_h^{i,j} * C_h^{i,j}
38.     h = h + 1;
39. end
40. refresh I.twu = 0 where I ∈ Thtw^{i,j} ;
41. begin for k = i to j                //2nd scan of db^{i,j}
42.     for each itemset I ∈ Thtw^{i,j} ∪ C_{h+1}^{i,j}
43.          I.u = I.u + UP_k (I);
44. end
45. for each itemset I ∈ Thtw^{i,j} ∪ C_{h+1}^{i,j}
46.     if (I.u ≧ s×P.count )
47.          Thu^{i,j} = Thu^{i,j} ∪ I;
48. end
49. return Thu^{i,j};
```

Figure 4. Incremental procedure of THUI-Mine.

3.2.2 *Incremental procedure of THUI-Mine*

As shown in Table 3, D^- indicates the unchanged portion of an ongoing transaction database. The deleted and added portions of an ongoing transaction database are denoted by Δ^- and Δ^+, respectively. It is worth mentioning that the sizes of Δ^+ and Δ^-, i.e., $|\Delta^+|$ and $|\Delta^-|$ respectively, are not required to be the same. The incremental procedure of *THUI-Mine* is devised to maintain temporal high utility itemsets efficiently and effectively. This procedure is shown in Figure 4. As mentioned before, this incremental step can also be divided into three sub-steps: (1) generating temporal high transaction-weighted utilization 2-itemsets in $D^- = db^{1,3} - \Delta^-$, (2) generating temporal high transaction-weighted utilization 2-itemsets in $db^{2,4} = D^- + \Delta^+$ and (3) scanning the database $db^{2,4}$ only once for the generation of all temporal high utility itemsets. Initially, after some update activities, old transactions Δ^- are removed from the database $db^{m,n}$ and new transactions Δ^+ are added (in Step 6). Note that $\Delta^- \subset db^{m,n}$. Denote the updated database as $db^{i,j}$. Note that $db^{i,j} = db^{m,n} - \Delta^- + \Delta^+$. We denote the unchanged transactions by $D^- = db^{m,n} - \Delta^- = db^{i,j} - \Delta+$. After loading $Thtw^{m,n}$ of $db^{m,n}$ into CF where $I \in Thtw^{m,n}$, we start the first sub-step, i.e., generating temporal high transaction-weighted utilization 2-itemsets in $D^- = db^{m,n} - \Delta^-$. This sub-step tries to reverse the cumulative processing which is described in the preprocessing procedure. From Step 8 to Step 16, we prune the occurrences of an itemset I, which appeared before partition P_i, by deleting the value I.twu where $I \in$ CF and I.start < i. Next, from Step 17 to Step 39, similarly to the cumulative processing in Section 3.2.1, the second sub-step generates generating temporal high transaction-weighted utilization 2-itemsets in $db^{i,j} = D^- + \Delta^+$ and employs the scan reduction technique to generate $C_{h+1}^{i,j}$. Finally, to generate temporal high utility itemsets, i.e., $Thu^{i,j}$, in the updated database, we scan $db^{i,j}$ for only once in the incremental procedure to find temporal high utility itemsets. Note that $Thtw^{i,j}$ is kept in main memory for the next generation of incremental mining.

4. EXPERIMENTAL EVALUATION

To evaluate the performance of *THUI-Mine*, we conducted experiments of using synthetic dataset generated via a randomized dataset generator provided by IBM Quest project [3]. However, the IBM Quest data generator only generates the quantity of 0 or 1 for each item in a transaction. In order to fit databases into the scenario of utility mining, we randomly generate the quantity of each item in each transaction, ranging from 1 to 5, as is similar to the model used in [14]. Utility tables are also synthetically created by assigning a utility value to each item randomly, ranging from 1 to 1000. Observed from real world databases that most items are in the low profit range, we generate the utility values using a log normal distribution, as is similar to the model used in [14]. Figure 5 shows the utility value distribution of 1000 items.

The simulation is implemented in C++ and conducted in a machine with 2.4GHz CPU and 1G memory. The main performance metrices used is execution time. We recorded the execution time that *THUI-Mine* spends in finding temporal high utility itemsets. The number of itemsets comparison of *THUI-Mine*, Two-Phase and MEU is presented in Section 4.1. Section 4.2 shows the performance comparison of *THUI-Mine* and Two-Phase. Results on scaleup experiments are presented in Section 4.3.

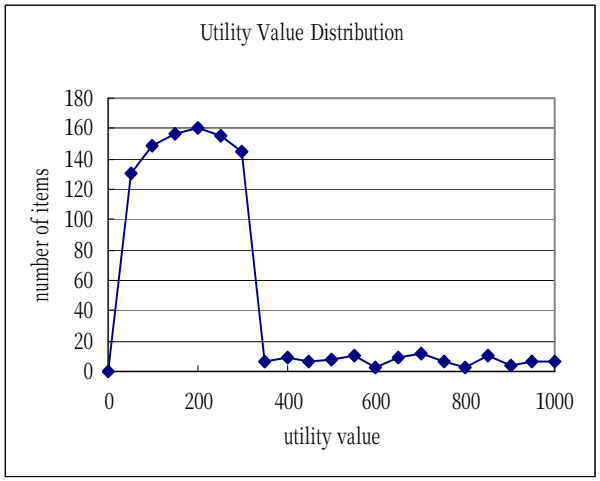


Figure 5. Utility value distribution in utility table.

4.1 Number of Generated Candidates

In this experiment, we compare the average number of candidates generated in the first database scan on the sliding windows and incremental transaction number d10K with different support values for *THUI-Mine*, Two-Phase [14] and MEU [19]. Without loss of generality, we set $|d| = |\Delta^+| = |\Delta^-|$ for simplicity. Thus, by denoting the original database as $db^{1,n}$ and the new mining database as $db^{i,j}$, we have $|db^{i,j}| = |db^{1,n} - \Delta^- + \Delta^+| = |D|$, where $\Delta^- = db^{1,i-1}$ and $\Delta^+ = db^{n+1,j}$. Table 4 and Table 5 show the average number of candidates generated by *THUI-Mine*, Two-Phase and MEU. The number of items is set at 1000, and the minimum utility threshold varies from 0.2% to 1%. The number of candidate itemsets generated by *THUI-Mine* at the first database scan decreases dramatically as the threshold goes up. Especially, when utility threshold is set as 1%, the number of candidate itemsets is 0 in database T10.I6.D100K.d10K where T denotes the average size of the transactions and I the average number of frequent itemsets. However, the number of candidates generated by Two-Phase is still very large and MEU is always 499,500 because it needs to process all combinations of 1000 items. *THUI-Mine* generates much fewer candidates compared to Two-Phase and MEU.

We obtain similar experimental results for different datasets. For example, only 118 candidate itemsets generated by *THUI-Mine*, but 183921 and 499500 candidate itemsets generated by Two-Phase and MEU when the utility threshold is set as 1% in dataset T20.I6.D100K.d10K. In the case of dataset T20.I6.D100K.d10K, more candidates are generated, because each transaction is longer than that in T10.I6.D100K.d10K. In overall, our algorithm *THUI-Mine* can always generate much fewer candidates compared to Two-Phase and MEU for various kinds of databases. Hence, *THUI-Mine* is verified to be very effective in pruning candidate itemsets to find temporal high utility itemsets.

Table 4. The average number of candidate itemsets generated by *THUI-Mine*, Two-Phase and MEU after the first scan on database T10.I6.D100K.d10K.

Databases / Threshold	T10.I6.D100K.d10K		
	THUI-Mine	Two-Phase	MEU
0.2%	3433	361675	499500
0.3%	666	303810	499500
0.4%	161	258840	499500
0.6%	7	182710	499500
0.8%	1	129286	499500
1%	0	91378	499500

Table 5. The average number of candidate itemsets generated by *THUI-Mine*, Two-Phase and MEU after the first scan on database T20.I6.D100K.d10K.

Databases / Threshold	T20.I6.D100K.d10K		
	THUI-Mine	Two-Phase	MEU
0.2%	27357	401856	499500
0.3%	11659	371953	499500
0.4%	5389	337431	499500
0.6%	1364	278631	499500
0.8%	371	229503	499500
1%	118	183921	499500

4.2 Evaluation of Execution Efficiency

In this experiment, we show only the relative performance of Two-phase and *THUI-Mine* since MEU spends much higher execution time and becomes incomparable. Figure 6 and Figure 7 show the execution times for the two algorithms as the minimum utility threshold is decreased from 1% to 0.2%. It is observed that when the minimum utility threshold is high, there are only a limited number of high utility itemsets produced. However, as the minimum utility threshold decreases, the performance difference becomes prominent in that *THUI-Mine* significantly outperforms Two-Phase. As shown in Figure 6 and Figure 7, *THUI-Mine* leads to prominent performance improvement for different average sizes of transaction. Explicitly, *THUI-Mine* is in orders of magnitude faster than Two-Phase, and the margin grows as the minimum utility threshold decreases. We could observe that *THUI-Mine* spends fewer times than Two-Phase with high stability for finding temporal high utility itemsets. This is because Two-Phase algorithm produces more candidate itemsets and needs more database scans to find high utility itemsets than our *THUI-Mine* algorithm. To measure how much execution time could be reduced substantially in using *THUI-Mine* compared to Two-Phase algorithm, we define the Improvement Ratio as follows:

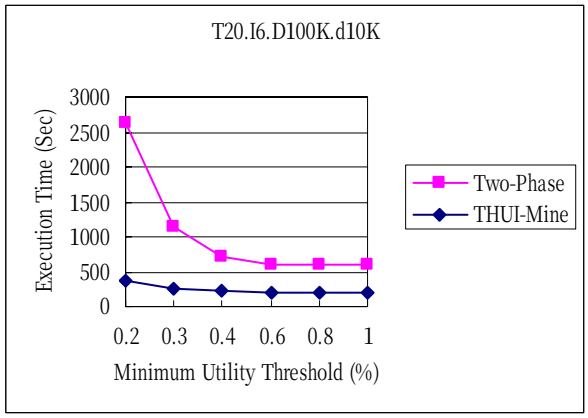


Figure 6. Execution time for Two-Phase and THUI on T20.I6.D100K.d10K.

Improvement Ratio = (execution time of Two-Phase − execution time of THUI-Mine) / (execution time of Two-Phase)

From Figure 6, we get that the Improvement Ratio is about 85.6% with the threshold set as 0.2%. In Figure 7, the average improvement is about 67% with minimum utility threshold varied from 0.2% to 1%. Obviously, *THUI-Mine* reduces substantially the time in finding high utility itemsets. Moreover, the high utility itemsets obtained by Two-Phase are not suitable for applications in data streams since Two-Phase needs more database scans and execution times in finding high utility itemsets by the time change. Hence, *THUI-Mine* meets the requirements of high efficiency in terms of execution time for data stream mining.

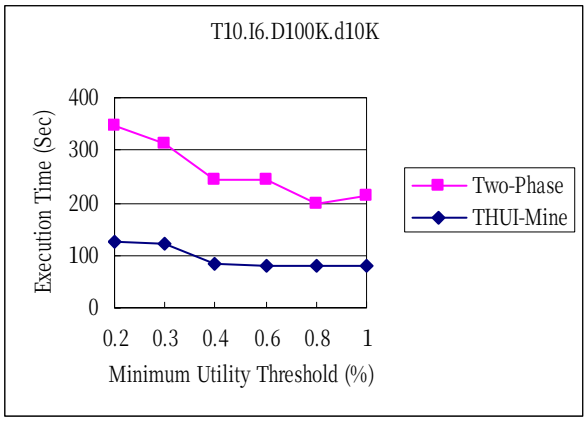


Figure 7. Execution time for Two-Phase and THUI on T10.I6.D100K.d10K.

4.3 Scaleup on Incremental Mining

In this experiment, we investigate the effects of varying incremental transaction size on the execution time of mining results. To further understand the impact of |d| on the relative performance of algorithms *THUI-Mine* and Two-Phase algorithms, we conduct the scaleup experiments which is similar in [12] for both *THUI-Mine* and Two-Phase with minimum support

thresholds being varied as 0.2% and 0.4 %, respectively. Figure 8 shows the experimental results where the value in y-axis corresponds to the ratio of the execution time of *THUI-Mine* to that of Two-Phase. Figure 8 shows the execution-time-ratio for different values of |d|. It can be seen that the execution-time ratio keeps stable with the growth of the incremental transaction number |d| since the size of |d| has little influence on the performance of *THUI-Mine*. Moreover, the execution time ratio of the scaleup experiments with minimum support thresholds varied from 0.6% to 1% keeps still around 0.4%. This implies that the advantage of *THUI-Mine* over Two-Phase is stable and less execution times is taken as the amount of incremental portion increases. This result also indicates that *THUI-Mine* fits for mining data streams with large transaction size.

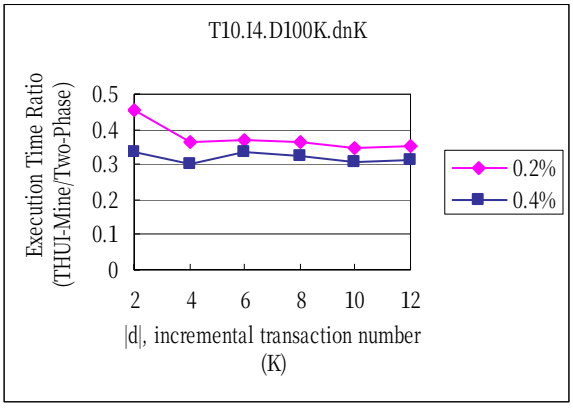


Figure 8. Scaleup performance with the execution time ratio between THUI and Two-Phase.

5. CONCLUSIONS

In this paper, we addressed the problem of discovering temporal high utility itemsets in data streams, i.e., the itemsets that are large than threshold in current time window of data stream. We propose a new approach, namely *THUI-Mine*, which can discover temporal high utility itemsets from data streams efficiently and effectively. The novel contribution of *THUI-Mine* is that it can effectively identify the temporal high utility itemsets with less temporal high transaction-weighted utilization 2-itemsets such that the execution time can be reduced efficiently in mining all high utility itemsets in data streams. In this way, the process of discovering all temporal high utility itemsets under all time windows of data streams can be achieved effectively with limited memory space, less candidate itemsets and CPU I/O. This meets the critical requirements of time and space efficiency for mining data streams.

The experimental results show that *THUI-Mine* can find the temporal high utility itemsets with higher performance by generating less candidate itemsets compared to other algorithms under different experimental conditions. Moreover, it performs scalable in terms of execution time under large databases. Hence, *THUI-Mine* is promising for mining temporal high utility itemsets in data streams. For future work, we would extend the concepts proposed in this work to discover other interesting patterns in data streams like utility item with negative profit.

ACKNOWLEDGMENTS

This research was supported in part by Ministry of Economic Affairs, R.O.C., under grant no. 92-EC-17-A-02-S1-024.

REFERENCES

[1] Agrawal, R., Imielinski, T., and Swami, A. Mining association rules between sets of items in large databases. In Proceedings of 1993 ACM SIGMOD Intl. Conf. on Management of Data, pages 207--216, Washington, D. C., May 1993.

[2] Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., and Verkamo, A. I. Fast discovery of association rules. In book Advances in Knowledge Discovery and Data Mining, pages 307--328. AAAI/MIT Press, 1996.

[3] Agrawal, R., and Srikant, R. Mining Sequential Patterns. *Proceedings of the 11th International Conference on Data Engineering*, pages 3-14, March 1995.

[4] Ayn, N.F., Tansel, A.U., and Arun, E. An efficient algorithm to update large itemsets with early pruning. Technical Report BU-CEIS-9908 Dept of CEIS Bilkent Uniiversity, June 1999.

[5] Ayn, N.F., Tansel, A.U., and Arun, E. An efficient algorithm to update large itemsets with early pruning. Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, August 1999.

[6] Bettini, C., Wang, X. S., and Jajodia, S. Testing complex temporal relationships involving multiple granularities and its application to data mining. In *Proc. of the 15th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, 1996, Montreal, Canada*, pages 68–78. ACM Press, 1996.

[7] Chan, R., Yang, Q., and Shen, Y. Mining high utility Itemsets. *Proc. of IEEE ICDM*, Florida, 2003.

[8] Cheung, D., Han, J., Ng, V., and Wong, C.Y. Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique. Proc. of 1996 Int'l Conf. on Data Engineering, pages 106—114, February 1996.

[9] Cheung, D., Lee, S.D., and Kao. B., A General Incremental Technique for Updating Discovered Association Rules. Proc. International Conference On Database Systems For Advanced Applications, April 1997.

[10] Chi, Y., Wang, H., Yu, P. S., and Richard, R. Muntz: Moment: Maintaining Closed Frequent Itemsets over a Stream Sliding Window Proceedings of the 2004 IEEE International Conference on Data Mining (ICDM'04).

[11] Das, G., Lin, K. I., Mannila, H., Renganathan G., and Smyth, P. Rule Discovery from Time Series. *Proceedings of the 4th ACM SIGKDD*, pages 16—22, August 1998.

[12] Lee, C. H., Lin, C. R., and Chen, M. S. Sliding-window filtering: An efficient algorithm for incremental mining. In *Intl. Conf. on Information and Knowledge Management* (CIKM01), pages 263－270, November 2001.

[13] Lin, J. L., and Dunham. M. H. Mining Association Rules: Anti-Skew Algorithms. Proc. of 1998 Int'l Conf. on Data Engineering, pages 486—493, 1998.

[14] Liu, Y., Liao, W., and Choudhary, A. A Fast High Utility Itemsets Mining Algorithm. In *Proceedings of the Utility-Based Data Mining Workshop*, August 2005.

[15] Park, J. S., Chen, M. S., and Yu. P. S. Using a Hash-Based Method with Transaction Trimming for Mining Association Rules. IEEE Transactions on Knowledge and Data Engineering, 9(5):813—825, October 1997.

[16] Savasere, A., Omiecinski, E., and Navathe, S. An Efficient Algorithm for Mining Association Rules in Large Databases. Proc. of the 21th International Conference on Very Large Data Bases, pages 432—444, September 1995.

[17] Teng, W. G., Chen, M. S., and Yu, P. S. A Regression-Based Temporal Pattern Mining Scheme for Data Streams. Proceedings of the 29th International Conference on Very Large Data Bases, pages 93—104, September 2003.

[18] Teng, W. G., Chen, M. S., and Yu, P. S. Resource-Aware Mining with Variable Granularities in Data Streams. SDM 2004.

[19] Yao, H., Hamilton, H. J., and Butz, C. J. A Foundational Approach to Mining Itemset Utilities from Databases. *Proc. of the4th SIAM International Conference on Data Mining*, Florida, USA, 2004.