Fast and Memory Efficient Mining of High Utility Itemsets in Data Streams.

CONFERENCE PAPER · JANUARY 2008

DOI: 10.1109/ICDM.2008.107 · Source: DBLP

5 AUTHORS, INCLUDING:

Yi-Cheng Chen
National Chiao Tung University
10 PUBLICATIONS **36** CITATIONS
SEE PROFILE

Suh-Yin Lee
National Chiao Tung University
135 PUBLICATIONS **1,273** CITATIONS
SEE PROFILE

Fast and Memory Efficient Mining of High Utility Itemsets in Data Streams

Hua-Fu Li[1], Hsin-Yun Huang[2], Yi-Cheng Chen[2], and Yu-Jiun Liu[2], and Suh-Yin Lee[2]
[1]*Department of Computer Science, Kainan University, Taoyuan, Taiwan*
[2]*Department of Computer Science, National Chiao-Tung University, Hsinchu, Taiwan*
E-mail: hfli@mail.knu.edu.tw; sylee@csie.nctu.edu.tw

Abstract

Efficient mining of high utility itemsets has become one of the most interesting data mining tasks with broad applications. In this paper, we proposed two efficient one-pass algorithms, ***MHUI-BIT*** *and* ***MHUI-TID****, for mining high utility itemsets from data streams within a transaction-sensitive sliding window. Two effective representations of item information and an extended lexicographical tree-based summary data structure are developed to improve the efficiency of mining high utility itemsets. Experimental results show that the proposed algorithms outperform than the existing algorithms for mining high utility itemsets from data streams.*

1. Introduction

Association rule mining (ARM) has been widely studied over the last decade. ARM model treats all items in a large database by only considering whether an item appeared in a transaction or not. The count of an itemset is not a sufficient indicator of interestingness. It only reflects the number of transactions in a large database that contains the itemset. However, it does not reveal the utility of an itemset. The utility can be measured in terms of cost, profit or other expressions of user preferences. Moreover, frequent itemsets may only contribute a small portion of the overall profit, whereas non-frequent itemsets may contribute a large portion of profit. Therefore, a new model of ARM, i.e., *utility mining model* [5], is proposed to address the limitation of traditional ARM. Based on the utility mining model, utility is a measure of how useful or profitable an itemset X is. The utility of an itemset X, i.e., $u(X)$, is the sum of the utilities of itemset X in all the transactions containing X. An itemset X is called a *high utility itemset* if and only if $u(X) \geq min\_utility$, where *min_utility* is a user-defined minimum utility threshold. Therefore, the goal of utility mining is to find a set of high utility itemsets from a large transaction database.

Formal definition of utility mining, a theoretical model, called MEU, and a unified framework were proposed by Yao et al. [10, 13]. In this work, the utility is defined as the combination of utility information in each transaction and additional resources. An efficient utility mining algorithm, called Two-Phase, was proposed by Liu et al. [11]. The basic idea of Two-Phase algorithm is based on the MEU. But Two-Phase algorithm not only can prune down the number of candidate itemsets, but also can find a complete set of high utility itemsets.

A *data stream* is an infinite sequence of data elements continuously arrived at a rapid rate [2, 4]. Frequent pattern mining is one of interesting problems of mining data streams [3, 6, 7, 8, 9]. However, less work were proposed for mining high utility itemsets from data streams. Tseng et al. [12] proposed the first method, called THUI-Mine, for mining temporal high utility itemsets from data streams. In the framework of THUI-Mine, a database is divided into a sequence of partitions. In the first scan of database, THUI-Mine employs a filtering threshold in each partition to generate a progressive set of itemsets. Then, it uses database reduction method to generate a set of candidate k-itemsets, where $k \geq 2$. Finally, it needs one more scan over the database to find a set of high utility itemsets from these candidate k-itemsets. There are two problems of THUI-Mine algorithm: *a lot of false candidate itemsets* and *huge memory requirement*. Therefore, in this paper, we propose two efficient algorithms for mining high utility itemsets from data streams. Experiments show that the proposed algorithms outperform the THUI-Mine algorithm.

The remainder of this paper is organized as follows. The problem is defined in Section 2. The proposed algorithms are described in Section 3. Section 4 discusses the experimental results. Finally, we conclude the work in Section 5.

2. Preliminaries

2.1. Problem Definition

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of n distinct literals called *items*. An **itemset** is a non-empty set of items. An itemset $X = (i_1, i_2, \ldots, i_k)$ with k items is referred to as k-itemset,

1550-4786/08 $25.00 © 2008 IEEE
DOI 10.1109/ICDM.2008.107

and the value k is called the **length** of X, and $i_j \in I$ for j = 1, …, n. A **transaction** T = <TID, (i_1, i_2, …, i_k)> consists of a transaction identifier (TID) and a set of items (i_1, i_2, …, i_k), where $i_j \in I, j = 1, 2, …, k$. A **data stream** DS = {T_1, T_2, …, T_m} is an infinite sequence of transactions, where m is the TID of latest incoming transaction. The **transaction-sensitive sliding window** (TransSW) of DS is a window that slides forward for every transaction [8]. The window at each slide has a fixed number, w, of transactions, and w is the **size** of the TransSW. Hence, *current transaction-sensitive sliding window* **CurTransSW** (or $\mathbf{TransSW_{N-w+1}}$) = [T_{N-w+1}, T_{N-w+2}, …, T_N], where the index ($N-w+1$) is the window identifier of current TransSW.

Example 1 An example data stream DS, as shown in Figure 1, is composed of four consecutive TransSWs, i.e., $TransSW_1$=[T_1, T_2, …, T_9], $TransSW_2$ = [T_2, T_3, …, T_{10}], $TransSW_3$=[T_3, T_4, …, T_{11}] and $TransSW_4$= [T_4, T_5, …, T_{12}], when the window size, i.e., w = 9, is given.

In the typical framework of frequent itemsets mining, the quantity of item purchased of each transaction is 1 or 0. However, in the framework of high utility itemsets mining, the quantity of item purchased is an arbitrary number. The *quantity* is called **local transaction utility (LTU).** The LTU, denoted as $\boldsymbol{o(i_p, T_q)}$, represents the quantity of item i_p in the transaction T_q. For example, $o(a, T_3)$ = 12 and $o(c, T_1)$ = 26 in Figure 1.

The **external utility (EU)**, i.e., *profit*, of an item i_p, denoted as $\boldsymbol{u(i_p)}$ is the value associated with item i_p in the utility table. For example, $u(a)$ = 3, $u(b)$ = 10 and $u(c)$ = 1. The **utility of an item i_p in transaction T_q**, denoted as $\boldsymbol{u(i_p, T_q)}$, is defined as $\boldsymbol{o(i_p, T_q) \times u(i_p)}$. For example, the utility of item a in transaction T_3 is 36, i.e., $u(a, T_3) = o(a, T_3) \times u(a) = 12 * 3 = 36$. The **utility of an itemset X in transaction** T_q, denoted as $u(X, Tq)$, is defined as $\sum_{i_p \in X} u(i_p, T_q)$, where $X = (i_1, i_2, …, i_k)$ is a k-itemset and $X \subseteq T_q$. For example, the utility of 2-itemset (ce) in transaction T_1 is 31, i.e., $u(ce, T_1) = u(c, T_1) + u(e, T_1) = 26 * 1 + 1 * 5 = 31$, and utility of 3-itemset (abe) in transaction T_6 is 48, i.e., $u(abe, T_6) = u(a, T_6) + u(b, T_6) + u(e, T_6) = 1 * 3 + 4 * 10 + 1 * 5 = 48$.

The **utility of an itemset X in TransSW**, denoted as $u(X) = \sum_{T_q \in D \wedge X \subseteq T_q} u(X, T_q)$, is the sum of the utilities of X in all the transactions of TransSW containing X as a subset. For example, the utility of 2-itemset (bd) in TransSW is 146, i.e., $u(bd) = u(bd, T_2) + u(bd, T_4) + u(bd, T_8) = 66 + 52 + 28 = 146$, when the size of TransSW is 9.

An itemset X is called a **high utility itemset** if and only if $u(X) \geq min\_utility$. For example, 2-itemset (bd) is a high utility itemset in $TransSW_1$ since $u(bd) = 146 \geq min\_utility$ if $min\_utility$ is 120 in $TransSW_1$.

Problem Statement Given a data stream DS, the size w of a transaction-sensitive sliding window $TransSW$, and a user-defined minimal utility threshold $min\_utility$, the task of this paper is to mine the set of high utility itemsets efficiently from current sliding window.

TID	Items with quantity (i:q)
T_1	(c:26), (e:1)
T_2	(b:6), (d:1)
T_3	(a:12), (d:1)
T_4	(b:1), (d:7)
T_5	(c:12), (e:2)
T_6	(a:1), (b:4), (e:1)
T_7	(b:10), (e:1)
T_8	(a:1), (b:1), (c:1), (d:3), (e:1)
T_9	(a:2), (b:1), (c:27), (e:2)
T_{10}	(b:6), (c:2)
T_{11}	(b:3), (d:2)
T_{12}	(b:2), (c:1)

A data stream is formed by transactions arriving in series

(a) An Example Data Stream

Item	Profit (per unit)
a	3
b	10
c	1
d	6
e	5

(b) An Example Utility Table

Figure 1: An example data stream and its utility table

2.2 TWDC-Property

In the framework of Boolean frequent itemset mining algorithms [1, 3, 6, 7], **downward closure property**, i.e., *if an itemset is frequent then all its subsets must be frequent*, is usually used to mine all frequent itemsets from a large database. However, the downward closure property can not be used for mining high utility itemsets.

For example, the utility of 1-itemset (d) is 72 in $TransSW_1$, i.e., $u(d)$ = 72, in Figure 1. It is a low utility itemset but its superset 2-itemset (bd) is a high utility itemset since $u(bd) = 146 > min\_utility$ if $min\_utility$ is 120. Hence, we need new properties to mine high utility itemsets.

In this section, a property, called **Transaction Weighted Downward Closure Property (TWDC-Property)** [12], is used in our proposed algorithms to mine the set of high utility itemsets in streaming transactions with a transaction-sensitive sliding window.

Definition 1 The **transaction utility** of the transaction T_q, denoted as $tu(T_q)$, is the sum of the utilities of *all* items in T_q. For example, $tu(T_1) = u(c, T_1) + u(e, T_1) = 26*1+1*5 = 31$. For example, the transaction utility of each transaction in the example data stream of Figure 1 is given in Figure 3.

Definition 2 The **transaction-weighted utilization** of an itemset X, denoted as $twu(X)$, is the sum of the transaction utilities of all transactions containing X. For example, in $TransSW_1$ of Figure 1, the transaction-weighted utilization of 2-itemset (bd) is 146, i.e., $twu(bd) = tu(T_2) + tu(T_4) + tu(T_8) = 66 + 52 + 28 = 146$.

Definition 3 An itemset X is called a **high transaction-weighted utilization itemset (HTU-itemset)** if and only if $twu(X) \geq min\_utility$, where $min\_utility$ is a user-defined minimum utility threshold. For example, if $min\_utility$ is 120, the 2-itemset (bd) is a high transaction-weighted utilization itemset since $twu(bd)$ is 146 in $TransSW_1$ of Figure 1.

Property 1 (Transaction-Weighted Downward Closure Property) Let X be a k-itemset and Y be a (k-1)-

itemset such that $Y \subset X$. If X is a high transaction-weighted utilization itemset, Y is also a high transaction-weighted utilization itemset.

For example, let *min_utility* is 120 in $TransSW_1$ of Figure 1, since 3-itemset (*abe*) is a high transaction-weighted utilization itemset, its subsets, i.e., (*a*), (*b*), (*e*), (*ab*), (*ae*), (*be*), are also high transaction-weighted utilization itemsets.

3. Efficient Mining of High Utility Itemsets

In this section, two algorithms, MHUI-BIT and MHUI-TID, are proposed to mine a set of high utility itemsets from data streams. The proposed algorithms consist of two major components, i.e., *item information* (discussed in Section 3.1) and *a lexicographical tree-based summary data structure* based on item information (discussed in Section 3.2).

3.1 Bitvector and TIDlist Representations

The first major component of proposed algorithms is **item information**, i.e., *effective representations of items*. Two effective representations of item information, i.e., **Bitvector** and **TIDlist**, are developed and used in the proposed methods to restrict the number of candidates and to reduce the processing time and memory usage needed. In this paper, we proposed two algorithms to mine the set of high utility itemsets based on Bitvector and TIDlist, respectively. Note that the first proposed algorithm based on the Bitvector representation is called **MHUI-BIT** (**M**ining **H**igh **U**tility **I**temsets based on **BIT**vector). Moreover, based on TIDlist representation, the second proposed method is called **MHUI-TID** (**M**ining **H**igh **U**tility **I**temsets based on **TID**list).

Definition 4 For each item x in the current transaction-sensitive sliding window TransSW, a bit-sequence with w bits, denoted as **Bitvector**(x), is constructed. The construction process of Bitvector is described as follows. If the item x is in the i-th transaction of current TransSW, the i-th bit of **Bitvector**(x) is set to be 1; Otherwise, the bit is set to be 0.

Definition 5 For each item x in the current transaction-sensitive sliding window TransSW, a sorted list with at least w values, denoted as **TIDlist**(x), is constructed. The construction process of TIDlist is described as follows. If the item x is in the i-th transaction of current TransSW, the value i is stored in the TIDlist(x).

For example, the representations of Bitvector and TIDlist of each item of $TransSW_1$ and $TransSW_2$ are given in Figure 2. From this figure, we can find that item (a) appears in transactions T_3, T_6, T_8, and T_9 of $TransSW_1$. Hence, the Bitvector of item (*a*), i.e., **Bitvector**(*a*), and the TIDlist of item (*a*), i.e., **TIDlist**(*a*), are <001001011> and {3, 6, 8, 9} in $TransSW_1$, respectively. Furthermore, **Bitvector**(*a*) and **TIDlist**(*a*) within $TransSW_2$ are <010010110> and {2, 5, 7, 8}, respectively.

Both algorithms are composed of three phases, i.e., *window initialization phase*, *window sliding phase*, and *high utility itemsets generation phase*. These phases are discussed from Section 3.2 to Section 3.4, respectively.

Items	Bitvectors of $TransSW_1$	Bitvectors of $TransSW_2$
a	<001001011>	<010010110>
b	<010101111>	<101011111>
c	<100010011>	<000100111>
d	<011100010>	<111000100>
e	<100011111>	<000111110>

(a) Bitvector representations of items

Items	TIDlists of $TransSW_1$	TIDlists of $TransSW_2$
a	{3, 6, 8, 9}	{2, 5, 7, 8}
b	{2, 4, 6, 7, 8, 9}	{1, 3, 5, 6, 7, 8, 9}
c	{1, 5, 8, 9}	{4, 7, 8, 9}
d	{2, 3, 4, 8}	{1, 2, 3, 7}
e	{1, 5, 6, 7, 8, 9}	{4, 5, 6, 7, 8}

(b) TIDlist representations of items

Figure 2: Bitvector and TIDlist of items

TID	Transaction Utility	TID	Transaction Utility
T_1	31	T_7	105
T_2	66	T_8	37
T_3	42	T_9	53
T_4	52	T_{10}	62
T_5	22	T_{11}	42
T_6	48	T_{12}	21

Figure 3: Transaction utility of each transaction of Example 1

3.2 Window Initialization Phase

The first phase is *window initialization phase*. The phase is activated while the number of transactions generated so far in a data stream is less than or equal to a user-defined sliding window size w. In this phase, the item information, i.e., Bitvector and TIDlist, and transaction utility of each transaction within a current sliding window are generated.

For example, the representations of Bitvector and TIDlist of Example 1 are given in Figure 2 and the table of transaction utility of each transaction within sliding windows, $TransSW_1$ and $TransSW_2$, is shown in Figure 3.

While the sliding window is full, the proposed lexicographical tree-based summary data structure, called **LexTree-2HTU** (**lex**icographical **tree** with **2**-**HTU**-itemsets), based on item information is constructed. The procedure of building the LexTree-2HTU is described as follows.

First, a set of high transaction-weighted utilization 1-itemsets (*1-HTU-itemsets*) is generated by using item information and the transaction utility table. Then, a set of candidate 2-itemsets, i.e., C_2, are generated by combining the set of 1-HTU-itemsets. As each candidate is generated, its corresponding transaction-weighted utility is determined immediately by using item-information and the transaction utility table. Finally, 1-HTU-itemsets and 2-HTU-itemsets whose transaction-weighted utility are greater than or equal to the minimum utility are maintained in the proposed summary data structure LexTree-2HTU.

Note that in the MHUI-TID algorithm, the TIDlist of a candidate k-itemset is generated by joining the TIDlists of the two (k-1)-itemsets, where $k \geq 2$ and the set of two (k-1)-itemsets is a subset of this k-itemset. In MHUI-BIT

algorithm, the Bitvector of the two (k-1)-itemsets is generated by performing bitwise AND operation on the Bitvectors of the two (k-1)-itemsets.

For example, five items, a, b, c, d and e, are 1-HTU-itemsets in the window $TransSW_1$ of Example 1. First, four candidate 2-itemsets {ab, ac, ad, ae} are generated from 1-HTU-itemset a. In the framework of MHUI-TID algorithm, the TIDlist(ab) in $TransSW_1$ is {6, 8, 9} which is generated by intersecting TIDlist(a) = {3, 6, 8, 9} and TIDlist(b) = {2, 4, 6, 7, 8, 9}. In the framework of MHUI-BIT algorithm, the Bitvector(ab) in $TransSW_1$ is <000001011> and is generated by performing bitwise AND operation on Bitvector(a) = <001001011> and Bitvector(b) = <010101111>.

Next, the transaction-weighted utility of candidate 2-itemset ab, i.e., twu(ab), can be obtained by summarizing the corresponding transaction utilities. Hence, twu(ab) = $tu(T_6) + tu(T_8) + tu(T_9)$ from the transaction utility table as shown in Figure 3. Other candidates {ac, ad, ae} are verified in the same way. Finally, two 2-HTU-itemsets, i.e., (ab) and (ae), are maintained in the LexTree-2HTU with prefix a as shown in Figure 4. We use the same process to construct the sub-trees with prefixes b, c, d, and e. The result of LexTree-2HTU of $TransSW_1$ is given in Figure 5.

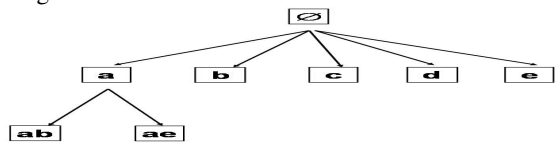

Figure 4: LexTree-2HTU after inserting all candidate 2-itemsets with a prefix item a

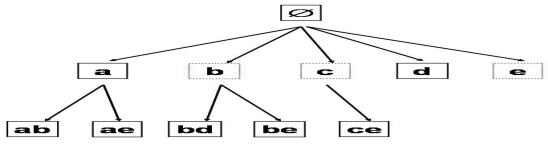

Figure 5: LexTree-2HTU after inserting all candidates of $TransSW_1$

3.3 Window Sliding Phase

The second phase of mining high utility itemsets, i.e., window sliding phase, is activated while the window is full and a new transaction arrives. In this phase, two operations are performed. The first operation is to update the item-information. The second one is to update the summary data structure LexTree-2HTU. Two operations are described as follows.

3.3.1 Update Item Information

In the framework of MHUI-TID algorithm, all TIDlists of items are sliding. The sliding means that the value of TIDlist is decreased by the number of dropped transactions. After sliding, new transactions are added. Hence, the TIDlists of items appeared in the new transaction are needed updated. For example, when window slides from $TransSW_1$ to $TransSW_2$, the oldest transaction T_1 is deleted and new incoming transaction T_{10} is added in the sliding window. At this time, the TIDlists of items in T_1 and T_{10} are updated. For instance, the TIDlists of items c and e of T_1 are changed from {1, 5, 8, 9} to {4, 7, 8} and from {1, 5, 6, 7, 8, 9} to {4, 5, 6, 7, 8}, respectively.

In MHUI-BIT algorithm, all Bitvectors of items are updated by performing bitwise AND operation for window sliding. For example, when window slides from $TransSW_1$ to $TransSW_2$, the oldest transaction T_1 is deleted and new incoming transaction T_{10} is added in the sliding window. At this time, the Bitvectors of items in T_1 and T_{10} are updated. For instance, the Bitvectors of items c and e of T_1 are changed from <100010011> to <000100110> and from <100011111> to <000111110>, respectively.

After updating item-information, we need to classify the items recorded in the dropped transaction and new incoming transaction into three types, i.e., *DeleteItem*, *InsertItem* and *IntersecItem*, for updating the proposed summary data structure LexTree-2HTU efficiently. The set of items in the dropped transaction is called **DeleteItems**. The set of items in the new incoming transaction is celled **InsertItems**. The set of items not only in the dropped transaction but also in the new incoming transaction is called **IntersecItems**. For example, item e is DeleteItem, item b is InsertItem, and item c is IntersecItem in Example 1.

3.3.2 Update LexTree-2HTU

The updating process of LexTree-2HTU is composed of three operations as described as follows.

(a) Item is a DeleteItem: Since the item is only in the oldest (dropped) transaction, the transaction-weighted utilities of its child nodes are less than or equal to that of previous window. That means its child nodes may be 2-HTU-itemsets in previous sliding window but are not 2-HTU-itemsets in the current sliding window. In this case, we check the child nodes of this item with its item-information based on different proposed algorithms and prune these child nodes while their transaction-weighted utilities are less than the user-defined minimum utility.

For example, since there are no potential candidate 2-itemsets generated from the item e (item e is a *leaf node* in this case) in previous sliding window, no itemsets need to be checked in the current LexTree-2HTU.

(b) Item is an InsertItem: Since the item is only in the new incoming transaction, the transaction-weighted utilities of its child nodes are greater than or equal to that of previous window. That means its child nodes may be not 2-HTU-itemsets in previous sliding window but may be 2-HTU-itemsets in the current sliding window. New candidate itemsets are generated from this new incoming transaction and their transaction-weighted utilities are

verified. If these new itemsets are high transaction-weighted utilization itemsets (HTU-itemsets), we insert these HTU-itemsets into the LexTree-2HTU immediately.

For example, item *b* is an InsertItem in Example 1. In the new incoming transaction T_{10}, only one candidate 2-itemsets (*bc*) is generated from item *b*, i.e., with prefix *b*. After verifying the transaction-weighted utility of candidate 2-itemset (*bc*), i.e., twu(*bc*) = tu(T_8) + tu(T_9) + tu(T_{10}) = 152 $>$ 120 (minimum utility), itemset (*bc*) is inserted into the LexTree-2HTU as a branch with a prefix item *b* as shown in Figure 5. Note that the existing child nodes of item *b*, i.e., (*bd*) and (*be*), are not necessary to check their transaction-weighted utilities, since the transaction-weighted utilities of itemsets (*bd*) and (*be*) are greater than the user-specified minimum utility and at least the same values as that of previous window. The result is given in Figure 6.

(c) Item is an IntersecItem: Since the item is not only in the dropped transaction but also in the new incoming transaction, original 2-HTU-itemset may be not a 2-HTU-itemset in current sliding window and vice versa. In this case, we check the transaction-weighted utilities of existing nodes appeared in the new incoming transaction to decide whether or not they need to be deleted from the LexTree-2HTU. Moreover, in order to decide whether or not new candidate itemsets are need to be inserted into the LexTree-2HTU, we also check the transaction-weighted utilities of these new candidates generated from the new transaction.

For example, item *c* is an IntersecItem in Example 1 since it appears in transactions T_1 = {(c: 26), (e: 1)} and T_{10} = {(b: 6), (c: 2)}. In this case, only one existing node, i.e., (*ce*), needs to be checked and no new candidate 2-itemsets need to be checked. After deleting the transaction T_1, the transaction-weighted utility of 2-HTU-itemset (*ce*), i.e., twu(*ce*), changes from 143 to 112. The new transaction-weighted utility of itemset (*ce*) is less than the user-defined minimum utility, i.e., 120. Hence, the node (*ce*) is deleted from the current LexTree-2HTU. The result is shown in Figure 7.

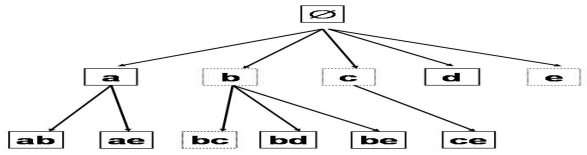

Figure 6: Updated LexTree-2HTU after processing the items in DeleteItem and InsertItem of Example 1

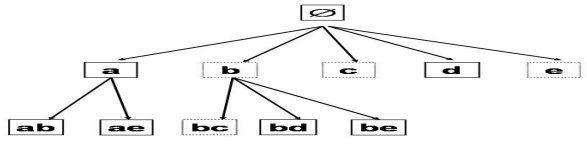

Figure 7: Updated LexTree-2HTU after processing the items in DeleteItem, InsertItem and IntersecItem of Example 1

Pattern Types	$TransSW_1$	$TransSW_2$
High Transaction-weighted Utilization Itemsets (generated after first scan)	a, b, c, d, e ab, ae, bd, be, ce, abc	a, b, c, d, e ab, ae, bc, bd, be, abc
High Utility Itemsets (generated after second scan)	b, bd, be	b, bd, be

Figure 8: Patterns generated after first and second scan of sliding window for $TransSW_1$ and $TransSW_2$ of Example 1

3.4 High Utility Itemsets Generation Phase

The phase of generating high utility itemsets is usually performed periodically or when it is needed. In this phase, the proposed algorithms use level-wise method to generate a set of candidate *k*-HTU-itemsets, C_k, from the previous pre-known (*k*-1)-HTU-itemsets, where $k > 2$. After that, we can immediately find the set of *k*-HTU-itemsets by using the item-information (Bitvector of MHUI-BIT algorithm and TIDlist of MHUI-TID algorithm). The candidate-generation-testing process stops when no candidates are generated.

For example, let the user-defined minimum utility is 120, an itemset *X* is a high utility if and only if u(*X*) ≥ 120. There are five 2-HTU-itemsets generated in $TransSW_1$ of Example 1. But only one candidate 3-itemset (*abe*) is generated by combining three 2-HTU-itemsets, i.e., (*ab*), (*ae*) and (*be*). After that, in MHUI-TID algorithm, TIDlist(*abe*) = {6, 7, 8} by joining TIDlist(*ab*) and TIDlist(*ae*). Hence, twu(*abe*) = tu(T_6) + tu(T_7) + tu(T_8) = 138 $>$ 120. Therefore, itemset (*abe*) is a 3-HTU-itemset.

Note that in MHUI-BIT algorithm, Bitvector(*abe*) = <000010110> by performing bitwise AND on Bitvector(*ab*) and Bitvector(*ae*). Because no more new candidates are generated, the candidate-generation-testing process stops. After all candidate HTU-itemsets are generated, one more scan of sliding window is needed to find high utility itemsets of $TransSW_1$. The result of high utility itemsets generated after first and second scan over each sliding window is shown in Figure 8.

4. Performance Evaluation

All the programs are implemented in C++ STL and compiled with Visual C++.NET compiler. All the programs are performed on AMD Athlon(tm) 64 Processor 3000+ 1.8GHz with 1GB memory and running on Windows XP system. All testing data was generated by the synthetic data generator provided by Agrawal et al in [1]. However, the IBM generator only generates the quantity of 0 or 1 for each item in a transaction. In order to adapt the databases into the scenario of utility mining, the quantity of each item and the utility of each item are randomly generated. In these experiments, the quantity of each item in each transaction, Q_{ip},, are generated randomly, ranging from 1 to 5. The utility of each item, U_{ip},, stored in the utility table is synthetically created by

assigning a utility value to each item randomly, ranging from 1 to 1,000. Observed from real world databases that most items are in the low profit range, the utility value generated using a log normal distribution, as is similar to the model used in the THUI-Mine algorithm [12].

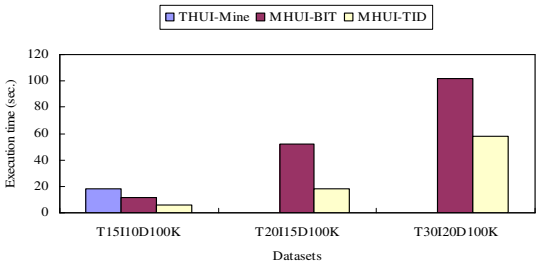

Figure 9: Execution time of algorithms MHUI-BIT, MHUI-TID, and THUI-Mine under various datasets

Minimum utility threshold (%)	THUI-Mine algorithm	MHUI-TID algorithm
0.9	218521	15
1.0	217162	8
2.0	138062	0
3.0	98203	0
4.0	87019	0
5.0	20885	0

Figure 10: Comparisons of the number of candidates

Three datasets, T15I10D100K, T20I15D100K and T30I20D100K, are used to compare the experimental results of these methods in Figure 9. The window size is fixed to 30,000. The partition size is fixed to 10,000. The minimum utility threshold is fixed to 1%. From this figure, we can see that THUI-Mine only runs successfully in the dataset T15I10D100K. However, the execution time of THUI-Mine can not be drawn in this picture since it needs much more execution time. From this figure, we can find that the relation of execution time needed is "MHUI-TID << MHUI-BIT << THUI-Mine".

The comparison of the number of candidates generated after 1st scan is given in Figure 10. From this figure, we can find that our algorithm generates less candidate itemsets than that of THUI-Mine. Hence, the proposed algorithms are memory efficient algorithms for mining high utility itemsets from data streams.

5. Conclusions

Mining of high utility itemsets is one of the most interesting research problems of data mining. In this paper, we proposed two efficient one-pass algorithms for mining a set of high utility itemsets form a transactional data stream. Experiments show that the proposed algorithms are efficient one-pass mining methods and outperform the existing algorithms for mining high utility itemsets from data streams. Future work includes mining top-k high utility itemsets from data streams and mining high utility itemsets from data streams with constraints.

Acknowledgements

The research is supported by National Science Council of R.O.C. under grant no. NSC96-2218-E-424-001- and NSC 95-2221-E-009-069-MY3.

References

[1] R. Agrawal and R. Srikant, Fast Algorithms for Mining Association Rules in Large Database, In Proc. of the 20th Intel. Conf. on Very Large Databases (VLDB), pp. 487- 499, 1994.

[2] Y. Zhu, D. Shasha, StatStream: Statistical Monitoring of Thousands of Data Stream in Real Time, In Proc. of the 28th Intel. Conf. on Very Large Databases (VLDB), pp. 358-369, 2002.

[3] G. Manku and R. Motwani, Approximate Frequency Counts over Data Streams, In Proc. of the 28th Intel. Conf. on Very Large Databases (VLDB), pp.346-357, 2002.

[4] L. Golab and M. T. Ozsu, Issues in Data Stream Management, In ACM SIGMOD Record, 32(2): pp. 5-14, 2003.

[5] R. Chan, Q. Yang, Y. D. Shen, Mining High utility Itemsets, In Proc. of the 3rd IEEE Intel. Conf. on Data Mining (ICDM), 2003.

[6] H.-F. Li, M.-K. Shan, and S.-Y. Lee, DSM-FI: An Efficient Algorithm for Mining Frequent Itemsets in Data Streams, Knowledge and Information Systems (KAIS), doi: 10.1007/s10115-007-0112-4.

[7] H.-F. Li and S.-Y. Lee, Mining Frequent Itemsets over Data Streams using Efficient Window Sliding Techniques, Expert Systems With Applications (ESWA), 39(3). doi: 10.1016/j.eswa.2007.11.061

[8] H.-F. Li, C.-C. Ho, and S.-Y. Lee, Incremental Updates of Closed Frequent Itemsets over Continuous Data Streams, Expert Systems with Applications (ESWA), 40(3). doi: 10.1016/ j.eswa.2007.12.054

[9] Y. Chi, H. Wang, P. S. Yu, R. Muntz, Moment: Maintaining Closed Frequent Itemsets over a Stream Sliding Window, In Proc. IEEE Intel. Conf. on Data Mining (ICDM), pp. 59-66, 2004.

[10] H. Yao, H. J. Hamilton, and C. J. Butz, A Foundational Approach to Mining Itemset Utilities from Databases, In Proc. of 4th SIAM Intel. Conf. on Data Mining (SDM), 2004.

[11] Y. Liu, W. Liao, and A. Choudhary, A Fast High Utility Itemsets Mining Algorithm, In Proc. of the ACM Intel. Conf. on Utility-Based Data Mining Workshop (UBDM), 2005.

[12] V. S. Tseng, C. J. Chu, and T. Liang, Efficient Mining of Temporal High Utility Itemsets from Data Streams, In Proc. of the ACM Intel. Conf. on Utility-Based Data Mining Workshop (UBDM), 2006.

[13] H. Yao, H. Hamilton and L. Geng, A Unified Framework for Utilty-Based Measures for Mining Itemsets, In Proc. of the ACM Intel. Conf. on Utility-Based Data Mining Workshop (UBDM), pp. 28-37, 2006.