An Efficient Approach for Mining High Utility Itemsets Over Data Streams

Show-Jane Yen and Yue-Shi Lee

Abstract *Mining frequent itemsets* only considers the number of the occurrences of the itemsets in the transaction database. *Mining high utility itemsets* considers the purchased quantities and the profits of the itemsets in the transactions, which the profitable products can be found. In addition, the transactions will continuously increase over time, such that the size of the database becomes larger and larger. Furthermore, the older transactions which cannot represent the current user behaviors also need to be removed. The environment to continuously add and remove transactions over time is called a data stream. When the transactions are added or deleted, the original high utility itemsets will be changed. The previous proposed algorithms for mining high utility itemsets over data streams need to rescan the original database and generate a large number of candidate high utility itemsets without using the previously discovered high utility itemsets. Therefore, this chapter proposes an approach for efficiently mining high utility itemsets over data streams. When the transactions are added into or removed from the transaction database, our algorithm does not need to scan the original transaction database and search from a large number of candidate itemsets. Experimental results also show that our algorithm outperforms the previous approaches.

Keywords Data mining · Knowledge discovery · High utility itemset · Frequent itemset · Closed itemset · Data stream · Large databases · Utility threshold · Mining algorithm · Information maintenance

S.-J. Yen (✉) · Y.-S. Lee
Department of Computer Science and Information Engineering,
Ming Chuan University, Taoyuan County, Taiwan
e-mail: sjyen@mail.mcu.edu.tw

W. Pedrycz and S.-M. Chen (eds.), *Data Science and Big Data: An Environment of Computational Intelligence*, Studies in Big Data 24,
DOI 10.1007/978-3-319-53474-9_7

1 Introduction

In this section, we first introduce some preliminaries for mining high utility itemsets [7]. Let I = {i1, i2,…, im} be the set of all the items. An itemset X is a subset of I and the length of X is the number of items contained in X. An itemset with length k is called a k-itemset. A transaction database D = {T1, T2,…, Tn} contains a set of transactions and each transaction has a unique transaction identifier (TID). Each transaction contains the items purchased in this transaction and their purchased quantities. The purchased quantity of item ip in a transaction Tq is denoted as o(ip, Tq). The utility of item ip in Tq is u(ip, Tq) = o(ip, Tq) × s(ip), in which s(ip) is the profit of item ip. The utility of an itemset X in Tq is the sum of the utilities of the items contained in X ⊆ Tq, which is shown in expression (1). If X ⊄ Tq, u(X, Tq) = 0. The utility of an itemset X in D is the sum of the utilities of X in all the transactions containing X, which is shown in expression (2).

The transaction utility (tu) of a transaction Tq is the sum of the utilities of the items in Tq, which is shown in expression (3). The total utility of the whole transaction database D is the sum of the transaction utilities of all the transactions in D. A utility threshold is a user specified percentage and a minimum utility (MU) can be obtained by multiplying total utility of D and the user-specified utility threshold. An itemset X is a high utility itemset if the utility of X in D is no less than the minimum utility.

$$u(X, T_q) = \sum_{i_p \in X \subseteq T_q} u(i_p, T_q) \tag{1}$$

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

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

For example, Table 1 is a transaction database, in which each integer number represents the purchased quantity for an item in a transaction. Table 2 is a Profit Table which records the profit for each item in Table 1. Suppose the user-specified utility threshold is 60%. Because the total utility of Table 1 is 224, the minimum utility is 226 * 60% = 134.4. The utility of itemset {D} is u ({D}) = 3 × 6 = 18 ≦

Table 1 A transaction database

Item / TID	A	B	C	D	E
T_1	0	0	16	0	1
T_2	0	6	0	1	1
T_3	2	0	1	0	0
T_4	0	10	0	1	1
T_5	1	0	0	1	1

Table 2 Profit table

Item	Profit ($) (Per Unit)
A	3
B	10
C	1
D	6
E	5

135, which is not a high utility itemset. The utility of itemset {BD} is u ({BD}) = (6 × 10 + 1 × 6) + (10 × 10 + 1 × 6) = 172 ≥ 135. Therefore, itemset {BD} is a high utility itemset.

For mining frequent itemset [1], all the subsets of a frequent itemset are frequent, that is, there is a downward closure property for frequent itemsets. However, the property is not available for high utility itemsets, since a subset of a high utility itemset may not be a high utility itemset. For the above example, itemset {BD} is a high utility itemset in Table 1, but its subset {D} is not a high utility itemset. Therefore, Liu et al. [7] proposed a *Two-Phase algorithm* for mining high utility itemsets. They defined the *transaction weighted utility* (twu) for an itemset X, which is shown in expression (4).

$$\text{twu}(X) = \sum_{X \subseteq T_q \in D} \text{tu}(T_q) \tag{4}$$

If the twu of an itemset is no less than MU, then the itemset is a high transaction weighted utility (HTWU) itemset. According to expression (4), the twu for an itemset X must be greater than or equal to the utility of X in D. Therefore, if X is a high utility itemset, then X must be a HTWU itemset. All the subsets of a HTWU itemset are also HTWU itemsets. Therefore, there is a downward closure property for HTWU itemsets. The first phase for the Two-Phase algorithm [7] is to find all the HTWU itemsets which are called candidate high utility itemsets by applying Apriori algorithm [1]. Two-Phase algorithm scans the database again to compute the utilities for all the candidate high utility itemsets and find high utility itemsets in the second phase.

Although some approaches [2, 7, 9, 15, 16, 18] have been proposed for mining high utility itemsets in a static transaction database, these approaches cannot efficiently discover high utility itmesets in a data stream environment, since they need to rescan the original database and re-discover all the high utility itemsets when some transactions are added into or removed from the database. In a data stream environment, the transactions are generated or removed in an extremely fast way. We need to immediately identify which itemsets can be turn out to be high utility itemsets, and vice versa. Besides, in this environment, we need to keep the information for all the itemsets, otherwise some high utility itemsets may be lost. However, the memory space is limited. It is very difficult to retain the utilities of all the itemsets in a large database.

Recently, some approaches [3, 6, 10, 14] have been proposed to find high utility itemsets in a data stream, which can be divided into Apriori-like [6, 14] and Tree-based approaches [3, 10]. However, these approaches just tried to find candidate high utility itemsets, that is HTWU itemsets. They still need to take a lot of time to rescan the original database and search for high utility itemsets from the large number of candidate itemsets without using the previous found information. Therefore, in this chapter, we propose an efficient algorithm HUIStream(mining High Utility Itemset in data Stream) for mining high utility itemsets in a data stream. When the transactions are added or deleted, our algorithms can just update HTWU itemsets according to the added or deleted transactions and directly calculate the utilities of HTWU itemsets without rescan the original database and search for high utility itemsets from the HTWU itemsets.

2 Related Work

The early approaches for mining frequent itemsets [1, 4, 12] are based on Apriori-like approach, which iteratively generate candidate $(k + 1)$-itemsets from the frequent k-itemsets ($k \geq 1$) and check if these candidate itemsets are frequent. However, in the cases of extremely large input sets or low minimum support threshold, the Apriori-like algorithms may suffer from two main problems of repeatedly scanning the database and searching from a large number of candidate itemsets.

Since Apriori-like algorithms require multiple database scans to calculate the number of occurrences of each itemset and record a large number of candidate itemsets, Tree-based algorithms [5, 11, 21] improve these disadvantages, which transform the original transaction database into an FP-tree and generate the frequent itemsets by recursively constructing the sub-trees according to the FP-Tree. Because all the transactions are recorded in a tree, Tree-based algorithms do not need multiple database scans and do not need to generate a large number of candidate itemsets.

Although Tree-based algorithms have been able to efficiently identify frequent itemsets from the transaction database, because of the number of the frequent itemsets may be very large, the execution time and memory usage would increase significantly. Therefore, some researchers have proposed the concept of closed itemsets [17, 20]. The number of the closed frequent itemsets is far less than the number of the frequent itemsets in a transaction database, and all the frequent itemsets can be derived from the frequent closed itemsets, so either memory usage or execution time for mining frequent closed itemsets is much less than that of mining frequent itemsets.

Liu et al. [7] proposed Two-Phase algorithm for mining high utility itemsets. Since the subset of a high utility itemsets may not be a high utility itemsets, that is, there is no downward closure property for high utility itemset, Liu et al. proposed the transaction weighted utility (twu) of an itemset to find out high utility itemsets.

In the first stage, Two-Phase algorithm applied Apriori algorithm [1] to find all the HTWU itemsets as the candidate itemsets, and then scans the transaction database to calculate the utility for each candidate itemset in order to identify which candidate itemsets are high utility itemsets. Although Two-Phase algorithm can find all the high utility itemsets from a transaction database, a large number of HTWU itemsets would be generated in the first phase, such that much time would be taken to search for high utility itemsets from these candidates in the second phase, since the twu of an itemset is much greater than the utility for the itemset.

Tseng et al. [14] proposed an algorithm THUI-Mine for mining high utility itemsets in a data stream, which only stores length two HTWU itemsets and applies Two-Phase algorithm to find all the HTWU itemsets. When a set of transactions is added, if there are new items in the added transactions, THUI-Mine will only determine whether the new items satisfy the utility threshold in the added transactions. If the items in the added transactions already exist in the original database, THUI-Mine will judge if the items are still HTWU items. Because THUI-Mine uses Two-Phase algorithm to re-mine the high utility itemsets, it still needs to take a lot of time to scan the database many times. HUP-HUI-DEL algorithm [8] also applies Two-Phase algorithm and only considers the transaction deletion. It still needs to generate a large number of candidate high utility itemsets and repeatedly scans the database to find high utility itemsets.

Li et al. [6] proposes MHUI algorithm, which discovers high utility itemsets in a specific sliding window. MHUI takes use of BITvector or TIDlist to store the transaction IDs in which each item is contained to avoid repeatedly scanning the database. MHUI stores length 2 HTWU itemsets in the structure LexTree-2HTU (Lexicographical Tree with 2-HTU itemset). When the transactions are added or deleted, MHUI generates all the length 2 itemsets from the added or deleted transactions and updates the structure LexTree-2HTU. MHUI uses level-wise method to generate all the HTWU itemsets from the length 2 HTWU itemsets, and re-sacan the database to find high utility itmesets.

Ahmed et al. [3] proposes a tree structure IHUP to stores the utility for each transaction and divides IHUP into three types according the order of the items which appear in the tree nodes: $IHUP_L$, $IHUP_{TF}$ and $IHUP_{TWU\text{-}}$. When a transaction is added, $IHUP_L$ stores each item of the transaction in the tree node according to the alphabetic order, but $IHUP_{TF}$ and $IHUP_{TWU}$ need to adjust the tree nodes to make sure that the items of each transaction are ordered by support and twu, respectively. IHUP needs to spend a large amount of memory space to store the whole database in a tree structure and applies FP-Growth algorithm [5] to repeatedly generate subtree structure. Finally, IHUP still needs to rescan the whole database to calculate the utility for each HTWU itmesets and generate high utility itemsets.

Yun and Ryang proposes HUPID-Growth algorithm [19] and SHU-Grow algorithm [13], respectively. HUPID-Growth scans the database once to construct HUPID-Tree and TIList and adjust the order of the items in the tree nodes to reduce the over-estimated value of the utility for each node in a path, that is to reduce the over-estimated utility for each itemsets. SHU-Grow uses the tree structure IHUP and stores the accumulated utility for each node when a set of transactions are

added. SHU-Grow applies the strategies of UP-Growth algorithm [16] to reduce the over-estimated utility and the number of the candidate high utility itemsets. UPID-Growth and SHU-Grow still apply FP-Growth algorithm to find HTWU itemsets and search for high utility itemsets from a large number of candidate high utility itemsets.

3 Mining High Utility Itemsets in a Data Stream

In this section, we first introduce the storage structure for our algorithm HUIStream. When a transaction is added into or deleted from the transaction database, HUIStream updates HTWU itemsets related to the added or deleted transaction, respectively. In the following, we propose two algorithms HUIStream+ and HUIStream− for maintaining the HTWU itemsets and generates all the high utility itemsets from the HTWU itemsets when a transaction is added and deleted, respectively.

In order to avoid rescanning the original database and searching from the candidate high utility itemsets when the transactions are added or deleted, we have the following definitions. An itmeset X is a closed twu itemset if there is no superset of X, which has the same twu as X. An itemset is a closed HTWU itemset if X is a closed twu itemset and the twu of X is no less than user-specified minimum utility. For any two itemsets X and Y ($X \subseteq Y$), if the twu of Y is the same as the twu of X and Y is not contained in any other itemset with the same twu as Y, then Y is the closure of X. For a closed HTWU itemset X, the proper subset of X, which has the same twu as X, is called the Equal TWU itemset of X, and X is the closure of the Equal TWU itemset.

In order to efficiently find the high utility itemsets in a data stream without information loss, HUIStream first determines which itemsets in the transaction are closed twu itemsets when a transaction is added or deleted. All the closed twu itemsets are recorded in a *Closed Table*, since the number of the closed itemsets is much less than the number of the itemsets and all the itemsets can be generated by the closed itemsets in a transaction database. There are three fields included in the Closed Table: *Cid* records the identification of each closed twu itemset; *CItemset* records the closed twu itemset with utility of each item in the closed twu itemset; *twu* records the twu of the closed twu itemset.

Table 3 shows the content of the Closed Table after the previous four transactions in Table 1 are added. There are five closed twu itemsets, in which the utility and twu of the closed twu itemset {E} with Cid 3 are 15 and 205, respectively. For each item, we use the table *Cid List* to record the Cids of the closed twu itemsets which contain the item. Table 4 shows the content of the Cid List after the previous four transactions in Table 1 are added, in which the field CidSet for item C is {1, 4, 5}, since item C is contained in the three closed twu itemsets with Cids 1, 4 and 5.

For example, the total utility of the previous four transactions in Table 1 is 210. If the utility threshold is 60%, that is the minimum utility is 210 * 60% = 126, then

Table 3 The closed table after processing the previous four transactions in Table 1

Cid	CItemset	twu
0	0	0
1	C:16, E:5	21
2	B:160, D:12, E:10	182
3	E:15	205
4	A:6, C:1	7
5	C:19	30

Table 4 The Cid list after processing the previous four transactions in Table 1

Item	CidSet
A	4
B	2
C	1, 4, 5
D	2
E	1, 2, 3

Table 5 Closed HTWU itemsets and their equal TWU itemsets

Closed HTWU itemset	Equal TWU itemset
{BDE}	{B}, {D}, {BD}, {BE}, {DE}, {BDE}
{E}	{E}

the closed HTWU itemsets are {BDE} and {E}. The Equal TWU itemsets for the two closed twu itemsets are shown in Table 5. The closed HTWU itemsets and their Equal TWU itemsets form the candidate high utility itemsets. HUIStream only needs to update the closed HTWU itemsets, that is, update the content of Closed Table and Cid List, and then the twu values of all the Equal TWU itemsets can be computed without rescanning the database.

3.1 The Algorithm HUIStream$^+$

In this subsection, we describe how HUIStream finds the closed twu itemsets which need to be updated and all the HTWU itemsets after adding a transaction. When a transaction T_{ADD} is added, the twu value will be increased just for T_{ADD} and the subsets of T_{ADD}. Therefore, HUIStream only considers whether T_{ADD} and the subsets of T_{ADD} are closed twu itemsets or not. If $X \subseteq T_{ADD}$ is a closed twu itemset before adding the transaction T_{ADD}, it must be a closed twu itemset after adding the transaction, because the twu value for the supersets ($\not\subset T_{ADD}$) of X would not be

changed [20]. Therefore, all the closed twu itemsets which need to be updated can be obtained by performing the intersections on T_{ADD} and all the closed twu itemsets in the Closed Table. However, the intersections of T_{ADD} and most of the closed twu itemsets would be empty. It will waste a lot of unnecessary time to intersect T_{ADD} with all the closed twu itemsets. In order to avoid that the intersection is empty, HUIStream identifies which closed twu itemsets contain some items in $T_{ADD} = \{i_1, i_2, \ldots, i_m\}$ from Cid List according to expression (5).

$$SET(\{TADD\}) = CidSet(i_1) \cup CidSet(i_2) \cup \ldots \cup CidSet(i_m) \quad (5)$$

The closed twu itemset X obtained by the intersection of each closed twu itemset Y with Cid in $SET(\{T_{ADD}\})$ and T_{ADD} need to be updated when a transaction T_{ADD} is added. The closed twu itemsets which need to be updated after adding transaction T_{ADD} are recorded in the table $Temp_{ADD}$, which includes the two fields: UItemset records the closed twu itemset X which needs to be updated; Closure_Id records the Cid of the closure of X before adding transaction T_{ADD}, that is, the Cid of itemset Y. If there is the same closed twu itemset X generated by the intersections of different closed twu itemsets and T_{ADD}, then the closure of X is the closed twu itemset with the largest twu value among the different closed twu itemsets. Because an added transaction T_{ADD} must be a closed twu itemset [20], T_{ADD} is recorded in $Temp_{ADD}$ and the corresponding Closure_Id is set to be 0, which represents that we cannot know the closure of T_{ADD} before adding the transaction so far. HUIStream can update the content of Closed Table according to the $Temp_{ADD}$.

For each record in $Temp_{ADD}$, HUIStream compares the itemset X in UItemset and the itemset Y with Cid in Closure_Id. If X and Y are the same, that is, X is a closed twu itemset before adding the transaction T_{ADD}, then the utility of each item in X is increased by adding the utility of the item in T_{ADD}, and the twu of X is increased by adding the tu of T_{ADD}. If X and Y are different, that is, X is not a closed twu itemset before adding the transaction T_{ADD} and turns out to be a closed twu itemset after adding the transaction, then HUIStream assigns X a unique Cid and adds it into The Closed Table and Cid List, in which the twu of X is the twu of Y plus the tu of T_{ADD} and the utility of each item is the utility of the item in Y in Closed Table plus the utility of the item in T_{ADD}, since the twu of X is equal to the twu of Y before adding the transaction T_{ADD}.

If itemset X is not a closed HTWU itemset before adding transaction T_{ADD}, but is closed HTWU itemsets after adding the transaction, then the Equal TWU itemsets of the Closure Y of X becomes the Equal TWU itemsets of X. If the Closure of X is not a closed HTWU Itemset, then the Equal TWU itmesets of X are the subsets of X, which have the same twu as X. HUIStream uses the following method to determine if the subset Z of X is an Equal TWU itemset of X: If the twu of X is the largest twu among all the itemsets with Cids in SET(Z) according to expression (5), then Z is an Equal TWU itemset of X, that is, X is the Closure of Z. If X is a closed twu itemset before adding the transaction, then HUIStream only needs to justify if X is a closed HTWU itemset after adding the transaction.

For example, suppose the utility threshold is 60% for Table 1. The Close Table and Cid List are shown in Tables 3 and 4 after adding the previous four transactions in Table 1. When the transaction T_5 is added, HUIStream records the itemset T_5 = {ADE} in the field UItemset of $Temp_{ADD}$ and set 0 to Closured_Id. Because SET(T_5) = CidSet(A) ∪ CidSet(D) ∪ CidSet(E) = {1, 2, 3, 4} according to expression (1), HUIStream performs the intersections of T_5 and the closed twu itemsets with Cids 1, 2, 3 and 4 from Closed Table, respectively. Firstly, because Cid 1 is itemset {CE} and {ADE} ∩ {CE} = {E}, HUIStream adds UItemset {E} and Closure_Id 1 in the $Temp_{ADD}$. Secondly, because Cid 2 is itemset {BDE} and {ADE} ∩ {BDE} = {DE}, HUIStream adds UItemset {DE} and Closure_Id 2 in the TempADD. Thirdly, Cid 3 is itemset {E} and {ADE} ∩ {E} = {E} has existed in TempADD. Because the twu of the closed twu itemset with Cid 3 is greater than the twu of the closed twu itemset with Cid 1, the corresponding Closure_Id of UItemset {E} is replaced with Cid 1. Finally, because Cid 4 is itemset {AC} and {ADE} ∩ {AC} = {A}, HUIStream adds UItemset {A} and Closure_Id 4 in the TempADD. After adding the transaction T_5, the $Temp_{ADD}$ is shown in Table 6.

HUIStream updates Closed Table and Cid List according to $Temp_{ADD}$. For Table 6, because the first UItemset {ADE} of $Temp_{ADD}$ is not in Closed Table and the corresponding Closure_Id is 0, which means that itemset {ADE} is not a closed twu itemset before adding transaction T_5, but turns out to be a closed twu itemset after adding transaction T_5, HUIStream adds the CItemset {A:3, D:6, E:5} with Cid 6 and twu = 3+6 + 5 = 14 into the Closed Table. HUIStream also inserts Cid 6 into Cid List for items A, D and E. Because the minimum utility is 135 after adding transaction T_5, itemset {ADE} is not a closed HTWU itemset.

For the second UItemset {E} and the corresponding Closure_Id 3 in Table 6, because Cid 3 in Closed Table (Table 3) is also {E}, which means that itemset {E} is a closed twu itemset before adding transaction T_5, the twu of {E} after adding the transaction T_5 is the twu of {E} in the Closed Table plus the tu of T5, that is 205 + 14 = 219. Because the utility of {E} is 5 in T_5, HUIStream updates the CItemset {E:15} in Table 3 as {E:20 (=15 + 5)}.

For the third UItemset {DE} and the corresponding Closure_Id 2 in Table 6, because Cid 2 in Closed Table (Table 3) is {BDE}, which means that itemset {DE} is not a closed twu itemset and the Closure of {DE} is {BDE} before adding transaction T_5, the twu of {DE} after adding transaction T5 is the twu of {BDE} in the Closed Table plus the tu of T_5, that is, 182 + 14 = 196. HUIStream adds the

Table 6 The TempADD after adding transaction T_5

UItemset	Closure_Id
{ADE}	0
{E}	3
{DE}	2
{A}	4

Table 7 The closed table after adding transaction T_5 in Table 1

Cid	CItemset	twu
0	0	0
1	C:16, E:5	21
2	B:160, D:12, E:10	182
3	E:20	219
4	A:6, C:1	7
5	C:17	28
6	A:3, D:6, E:5	14
7	D:18, E:15	196
8	A:9	21

Table 8 The Cid list after adding transaction T_5 in Table 1

Item	CidSet
A	4, 6, 8
B	2
C	1, 4, 5
D	2, 6, 7
E	1, 2, 3, 6, 7

Table 9 The closed HTWU itemsets after adding transaction T_5 in Table 1

Closed HTWU itemset	Equal TWU itemset
{BDE}	{B}, {BD}, {BE}, {BDE}
{E}	{E}
{DE}	{D}, {DE}

CItemset {D:18 (=12 + 6), E:15 (=10 + 5)} in the Closed Table and assigns the new closed twu itemset {DE} a Cid 7, which is added to Cid List for items D and E. Because the two itemsets {D} and {DE} are the Equal TWU itemsets of {BDE} before adding transaction T_5, and {D} and {DE} both are contained in {DE}, the two itemsets become the Equal TWU itemset of {DE} after adding transaction T_5.

For the fourth record in $Temp_{ADD}$, the UItemset is {A} and the corresponding Closure_Id is 4. Because Cid 4 in the Closed Table (Table 3) is {AC}, which means that itemset {DE} is not a closed twu itemset and the Closure of {A} is {AC} before adding transaction T_5. The twu of {A} after adding transaction T_5 is the twu of {AC} in the Closed Table plus the tu of T5, that is 7 + 14 = 21, which is not a closed HTWU itemset. HUIStream adds the CItemset {A:9 (=6 + 3)} in the Closed Table and assigns the new closed twu itemset {A} a Cid 8, which is added to Cid List for item A. After adding transaction T_5, the Closed Table and Cid List are shown in Table 7 and Table 8, and the closed HTWU itemsets and the corresponding Equal TWU itemsets are shown in Table 9.

3.2 The Algorithm HUIStream⁻

In this subsection, we describe how HUIStream finds the closed twu itemsets which need to be updated and all the HTWU itemsets after deleting a transaction. The itemsets need to be updated after deleting a transaction are the subsets of T_{DEL}, since the twu of the subsets of T_{DEL} would be decreased after deleting the transaction. Therefore, the closed twu itemsets which need to be updated can be obtained by performing the intersections on T_{DEL} and all the closed twu itemsets before deleting transaction T_{DEL}. In order to avoid that the intersection is empty, HUIStream only performs the intersections on T_{DEL} and the closed twu itemsets with Cids in SET({T_{DEL}}) according to expression (5) after deleting transaction TDEL. The closed twu itemsets which need to be updated after deleting a transaction are recorded in a table $Temp_{DEL}$, which includes the two fields: DItemset records the closed twu itemset X which needs to be updated; C1 records the Cid of X before deleting the transaction; C2 records the information which can be used to determine if X is still a closed twu itemset after deleting the transaction. Because a deleted transaction T_{DEL} is a closed twu itemset before the deletion [20], HUIStream firstly puts T_{DEL} in the first record of $Temp_{DEL}$, and sets the corresponding C1 and C2 to be 0.

Because the intersection of T_{DEL} and different closed twu itemsets S may obtain the same itemset X, the field C1 in $Temp_{DEL}$ records the Cid p of the closed twu itemset with the largest twu among all the closed twu itemsets in S. Cid p is the Cid of itemset X, since itemset X is a closed itemset before deleting the transaction T_{DEL}. Because itemset X may not be a closed twu itemset after deleting the transaction, the field C2 in $Temp_{DEL}$ records the Cid q of the closed twu itemset with the largest twu among all the closed twu itemsets in S except the closed twu itemset with Cid p. If the twu of Cid q is equal to the twu of Cid p minus the tu of T_{DEL}, which means that the itemset with Cid q has the same twu as X, then X is not a closed twu itemset any more after deleting the transaction. HUIStream updates the content of the Closed Table according to the Table $Temp_{DEL}$.

For each record with values X, p and q for DItemset, C1 and C2 in $Temp_{DEL}$, respectively, the twu values of the closed twu itemsets with Cid p and Cid q can be obtained from Closed Table. If the twu of X minus the tu of T_{DEL} is equal to 0, which means that X is not contained in any transaction after deleting T_{DEL}, then itemset X with Cid p is removed from the Closed Table. If itemset Y with Cid q is not itemset X and the twu of X minus the tu of T_{DEL} is equal to the twu of Y, then X is not a closed twu itemset after deleting transaction T_{DEL}, since Y is a superset of X and they have the same twu values.

If X is still a closed twu itemset after the deletion, then HUIStream updates the twu of X and the utility of each item in X in the Closed Table as follows: the updated twu of X is the twu of X minus the tu of T_{DEL} and the updated utility of each item in X is the utility of the item in X minus the utility of the item in T_{DEL}. If X is not a closed HTWU itemset before deleting the transaction but is a closed

Table 10 The $Temp_{DEL}$ after deleting the transaction T_1

DItemset	C1	C2
{CE}	1	0
{E}	3	7
{C}	5	4

HTWU itemset after the deletion, then HUIStream finds all the subsets of X, which have the same twu as X, that is all the Equal TWU itemsets of X.

For example, in Table 1, when the transaction T_1 = {CE} is deleted, HUIStream firstly puts {CE} in the field DItemset in $Temp_{DEL}$ and the corresponding C1 and C2 are set to be 0. Because SET(T1) = CidSet(C) ∪ CidSet(E) = {1, 2, 3, 4, 5, 6, 7} according to expression (1) and Cid List in Table 8, HUIStream performs the intersections on T_1 and the closed twu itemsets with Cids 1, 2, 3, 4, 5, 6 and 7 from Closed Table in Table 7, respectively. Firstly, Cid 1 is itemset {CE} and {CE} ∩ {CE} = {CE} which exists in $Temp_{DEL}$ and the corresponding C1 and C2 are 0 s. Because from Table 7, we can see that the twu of Cid 1 is greater than the twu of Cid 0, the Cid in C1 is changed to 1 and the Cid in C2 remains 0 for DItemset {CE} in the table $Temp_{DEL}$.

Secondly, because Cid 2 is itemset {BDE} and {CE} ∩ {BDE} = {E} which is not in $Temp_{DEL}$, the itemset {E} is added to $Temp_{DEL}$, and the corresponding C1 and C2 are set to 2 and 0, respectively. Thirdly, Cid 3 is itemset {E} and {CE} ∩ {E} = {E} has existed in $Temp_{DEL}$. Because the twu of Cid 3 is greater than the twu of Cid 2 and the twu of Cid 2 is greater than the twu of Cid 0, the Cid in C1 is replaced with 3 and the Cid in C2 is replaced with 2 for DItemset {E} in the table $Temp_{DEL}$. HUIStream continuously performs the intersections on T_1 and the closed twu itemsets with Cids 4, 5, 6, and 7, respectively, and updates the content of $Temp_{DEL}$. After deleting the transaction T_1, the $Temp_{DEL}$ is shown in Table 10.

HUIStream updates the content of Closed Table and Cid List according to TempDEL. For example, in Table 10, the first record in DItemset is {CE} and C1 is Cid 1. Because the twu of {CE} with Cid 1 in Table 7 minus the tu of T_1 is equal to 0, itemset {CE} is not a closed twu itemset after deleting transaction T_1. Therefore, HUIStream removes the information about {CE} from Closed Table and Cid List. The second record in DItemset is {E} and C1 is Cid 3. Because the twu of {E} with Cid 3 in Table 7 minus the tu of T_1 is equal to the twu of {DE} with Cid 7 in C2, itemset {E} is not a closed twu itemset after deleting transaction T_1, since there exists a superset {DE} of {E} and they have the same twu values. HUIStream removes the information about {E} from the Closed Table and Cid List, and moves all the Equal TWU itemsets of {E} to the Equal TWU itemsets of {DE} with Cid 7. There is the same situation with the second record for the third record in TempDEL. All the information about itemset {C} is removed from the Closed Table and Cid List. After deleting transaction T1 from Table 1, the Closed Table and Cid List are shown in Table 11 and Table 12, respectively, and the closed HTWU itemsets and their Equal TWU itemsets are shown in Table 13.

Table 11 The closed table after deleting the transaction T_1 from Table 1

Cid	CItemset	twu
0	0	0
2	B:160, D:12, E:10	182
4	A:6, C:1	7
6	A:3, D:6, E:5	14
7	D:18, E:15	196
8	A:9	21

Table 12 The Cid list after deleting the transaction T_1 from Table 1

Item	CidSet
A	4, 6, 8
B	2
C	4
D	2, 6, 7
E	2, 6, 7

Table 13 The closed HTWU itemsets and their equal TWU itemsetsafter deleting T_1

Closed HTWU itemset	Equal TWU itemset
{BDE}	{B}, {BD}, {BE}, {BDE}
{DE}	{D}, {E}, {DE}

Table 14 The candidate high utility itemsets and their utilities

Closed HTWU Itemset	Equal TWU itemsets
{B:160, D:12, E:10}	u({B}) = 160 u({BD}) = 172 u({BE}) = 170 u({BDE}) = 182
{D:18, E:15}	u({D}) = 18 u({E}) = 15 u({DE}) = 33

3.3 High Utility Itemset Generation

After processing the added and deleted transactions, all the Equal TWU itemsets of the closed HTWU itemsets are the candidate high utility itemsets. The utility of each Equal TWU itemset X for each closed HTWU itemset Y can be obtained by accumulating the utility of each item of X in Y from the Closed Table, and then all the high utility itemsets can be obtained without scanning the database.

For example, from Table 13, we can see that the itemsets {BDE} and {DE} are closed HTWU itemsets. For itemset {BDE} with Cid 2, the utility of {BDE} is 182(=B:160 + D:12 + E:10), which can be obtained from Closed Table in Table 11. The utility of the Equal TWU itemset {BD} of {BDE} is 172 (=B:160 + D:12), The utility of the Equal TWU itemset {BE} of {BDE} is 170 (=B:160 + E:10). All the candidate high utility itemsets and their utilities after deleting transaction T_1 are shown in Table 14, in which itemsets {B},{BD},{BE} and {BDE} are high utility itemsets.

4 Experimental Results

In this section, we evaluate the performance of our HUIStream algorithm and compare it with IHUP algorithm [3]. Our experiments are performed on Intel(R) Core(TM) 2 Quad CPU Q9400 @ 2.66 GHz with 4 GB RAM and running on Windows XP. The two algorithms are implemented in JAVA language.

We first generate two synthetic datasets T5I2D100 K and T5I4D100 K by using IBM Synthetic Data Generator [22], in which T is the average length of the transactions, I is the average size of maximal potentially frequent itemsets and D is the total number of the transactions. The number of distinct items is set to 1000. For the profit of each item, we use the log Normal Distribution [2, 7] and set the range of the profits between 0.01 and 10, which is shown in Fig. 1. The purchased quantity for an item in a transaction is randomly set to the number between 1 and 10.

Figure 2 and Fig. 3 show the execution time of IHUP and HUIStream, which the utility threshold is set to be 0.1%, the number of transactions is increased from 10 K to 100 K, and the size of sliding window is set to be 1 K and 10 K, respectively. From the experiments, we can see that HUIStream outperforms IHUP, and the performance gap increases as the number of transactions increases and the times of window size movements increases, since HUIStream only updates the closed twu

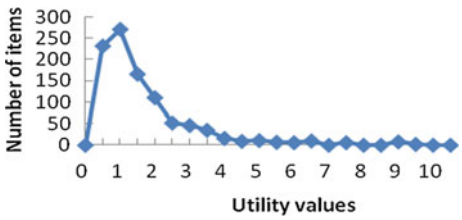

Fig. 1 Utility value distribution in utility table

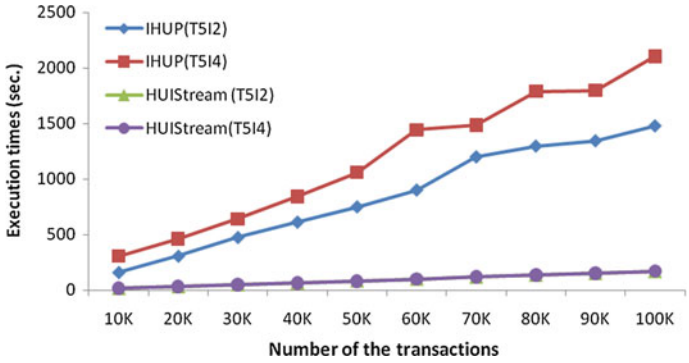

Fig. 2 The execution time for the two algorithms on window size = 1 K

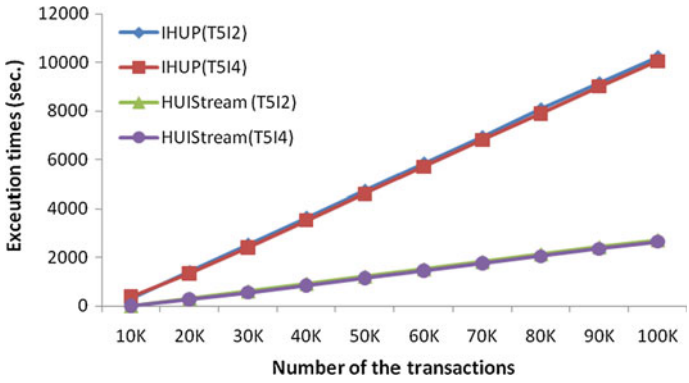

Fig. 3 The execution time for the two algorithms on window size = 10 K

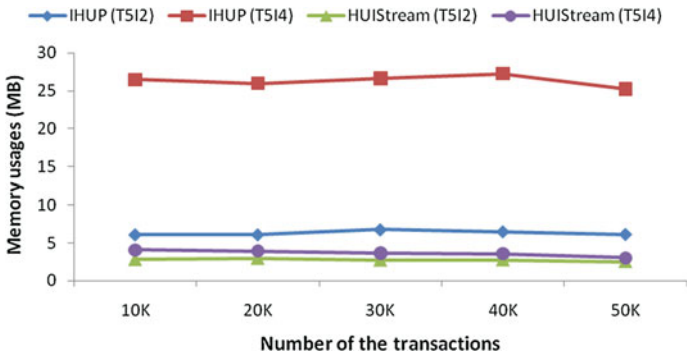

Fig. 4 The memory usages for the two algorithms on window size = 1 K

itemsets related to the added or deleted transactions. However, IHUP needs to re-mine HTWU itemsets by using FP-Growth algorithm [5] and rescan the database to find high utility itemsets from the HTWU itemsets.

Figures 4 and 5 show the memory usages for the two algorithms IHUP and HUIStream when the number of transactions is increased from 10 K to 50 K and the size of sliding window is set to be 1 K and 10 K, respectively, from which we can see that the memory usage for IHUP is significantly larger than that of HUI-Stream. This is because IHUP needs to recursively construct the subtrees for re-mining HTWU itemsets when a transactions are added or deleted, but HUI-Stream only needs to store and update the Closed Table and Cid List.

In the following experiments, we generate the two datasets T10I4D100K and T10I6D100K. The number of distinct items is set to 2000, and the utility threshold is set to be 0.1%. s Figures 6 and 7 show the execution time of IHUP and HUIStream, which the number of transactions is increased from 10 K to 50 K, and the size of sliding window is set to be 1 K and 10 K, respectively. From Fig. 6,

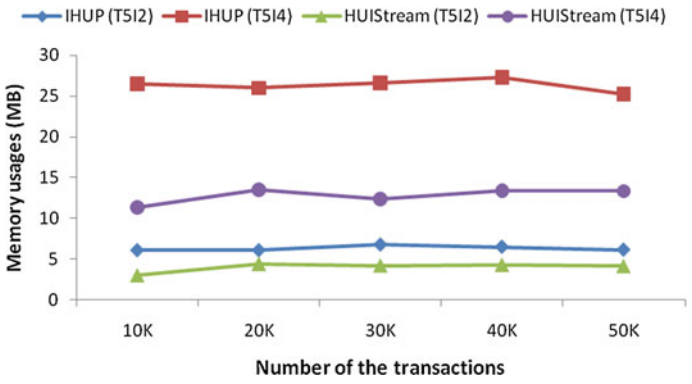

Fig. 5 The memory usages for the two algorithms on window size = 10 K

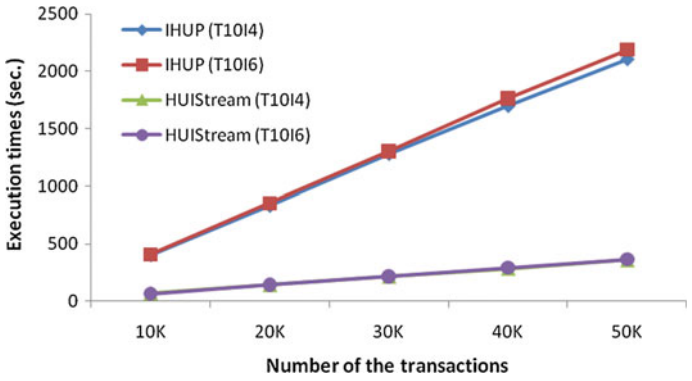

Fig. 6 The execution time for the two algorithms on window size = 1 K

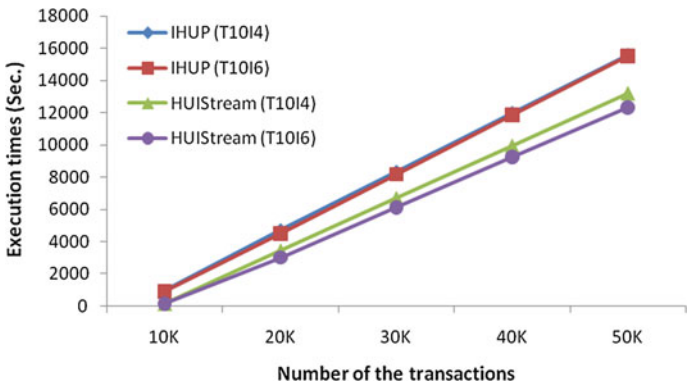

Fig. 7 The execution time for the two algorithms on window size = 10 K

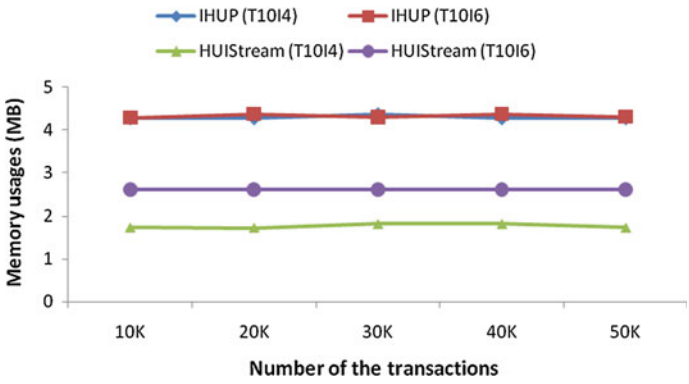

Fig. 8 The memory usages for the two algorithms on window size = 1 K

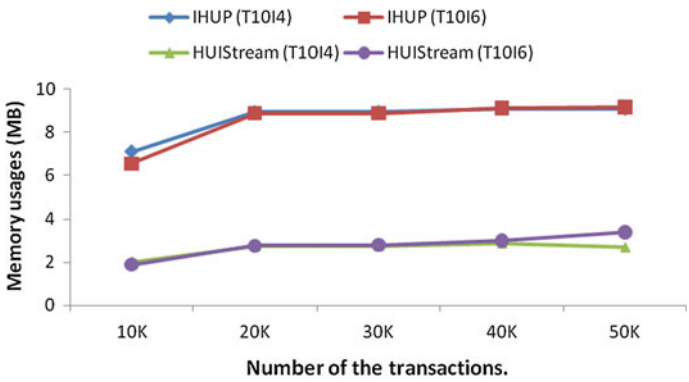

Fig. 9 The memory usages for the two algorithms on window size = 10 K

we can see that HUIStream outperforms IHUP and the performance gap increases as the number of transactions increases. Although the performance gaps are similar when the number of transactions increases from 10 K to 50 K, HUIStream still outperforms IHUP in Fig. 7. The memory usages for IHUP and HUIStream in this experiment are shown in Figs. 8 and 9, from which we can see that the memory usage for HUIStream still less than the memory usage for IHUP on the datasets with longer transaction size.

5 Conclusion

There are many previous approaches for mining high utility itemsets in a data stream. However, they all first need to generate a large number of candidate high utility itemsets and then scan the whole database to caculate the utility for each high

utility itemset. Although some approaches propose some strategies to reduce the number of the candidate high utility itemsets, the number of the candidates is still large when the size of the database is large. In order to avoid rescanning the database, some approaches store the whole database in a tree structure, but they also need to re-generate the candidate high utility itemsets when the transactions are added or deleted without using the information about previously discovered high utility itemsets.

In order to improve the performance of the previous approaches, we propose an algorithm HUIStream for mining high utility itemsets over a data stream. We take use of the concept of closed itemsets [20] and propose the definition of closed twu itemsets which can be used to derive the twu values of all the itemsets in the database. Because the number of the closed twu itemsets is much less than the number of the itemsets in the database, HUIStream only keeps all the closed twu itemsets, such that the twu values of all the itemsets in the database can be reserved. Therefore, our approach only needs to update the closed twu itemsets about the added or deleted transaction without any information loss when a transaction is added or deleted. According to the closed twu itemsets, HUIStream can directly obtain the high utility itemsets from the closed HTWU itemsets without rescanning the database. The experimental results also show that our HUIStream outperforms the other approaches.

References

1. Agrawal, R., Srikant, R.: Fast Algorithms for Mining Association Rules. In: Proceedings of 20th International Conference on Very Large Databases, Santiago, Chile, 487–499 (1994)
2. Ahmed, C. F., Tanbeer, S. K., Jeong B.S., Lee. Y.K.: An Efficient Candidate Pruning Technique for High Utility Pattern Mining. In: Proceedings of the 13th Pacific-Asia Conference on Knowledge Discovery and Data Mining, 749–756 (2009)
3. Ahmed, C. F., Tanbeer, S. K., Jeong, B. S., Lee, Y. K.: Efficient tree structures for high utility pattern mining in incremental databases. IEEE Transactions on Knowledge and Data Engineering, **21**(12), 1708–1721 (2009)
4. Brin, S., Motwani, R., Ullman, J., Tsur, S.: Dynamic itemset counting and implication rules for market basket data. In: Proceedings ACM SIGMOD International Conference on management of Data, 255–264 (1997)
5. Han, J., Pei, J., Yin, Y., Mao, R.: Mining frequent patterns without candidate generation: a frequent-pattern tree approach. Data Mining and Knowledge Discovery, **8**(1), 53–87 (2004)
6. Li, H.F., Huang, H.Y., Chen, Y.C., Liu, Y.J., Lee, S.Y.: Fast and memory efficient mining of high utility itemsets in data streams. In: Proceedings of the 8th IEEE International Conference on Data Mining, 881–886 (2008)
7. Liu, Y., Liao, W. K., Choudhary, A.: A Fast High Utility Itemsets Mining Algorithm. In: Proceedings of the International. Workshop on Utility-Based Data Mining, 90–99 (2005)
8. Lin, C. W., Lan, G. C., Hong, T. P.: Mining high utility itemsets for transaction deletion in a dynamic database. Intelligent Data Analysis **19**(1), 43–55 (2015)
9. Li, Y.C., Yeh, J.S., Chang, C.C.: Isolated Items Discarding Strategy for Discovering High Utility Itemsets. Data and Knowledge Engineering, **64**(1), 198–217 (2008)
10. Morteza, Z., Aijun, A.: Mining top-k high utility patterns over data streams. Information Sciences, **285**(1), 138–161 (2014)

11. Mohammad, E., Osmar, R. Z.: COFI approach for mining frequent itemsets revisited. In: Proceedings of the 9th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery, 70–75 (2004)
12. Park, J. S., Chen, M. S., Yu, P. S.: An Effective Hash-Based Algorithm for Mining Association Rules. ACM SIGMOD **24**(2), 175–186 (1995)
13. Ryang, H., Yun, U.: High utility pattern mining over data streams with sliding window technique. Expert Systems with Applications, **57**, 214–231(2016)
14. Tseng, S.M., Chu, C. J., Liang, T.: Efficient mining of temporal high utility itemsets from data streams. In: Proceedings of the ACM International Conference on Utility-Based Data Mining Workshop, 18–27 (2006)
15. Tseng, S.M., Shie, B.E., Philip Yu, S.: Efficient algorithms for mining high utility itemsets from transactional databases. IEEE Transactions on Knowledge and Data Engineering, **25**(8), 1772–1786 (2013)
16. Tseng, S.M., Wu, C.W., Shie, B.E., Philip Yu, S: UP-Growth: an efficient algorithm for high utility itemset mining. In: ACM SIGKDD, 253–262 (2010)
17. Wang, J., Han, J., Pei, J.: CLOSET+: Searching for the Best Strategies for Mining Frequent Closed Itemsets. In: Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 236–245 (2003)
18. Yen, S.J., Chen, C.C., Lee, Y.S.: A fast algorithm for mining high utility Itemsets. In: Proceedings of International Workshop on Behavior Informatics, joint with the 15th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), 171–182 (2011)
19. Yun, U., Ryang, H.: Incremental high utility pattern mining with static and dynamic databases. Applied Intelligence, **42**(2), 323–352(2015)
20. Yen, S.J., Wu, C.W., Lee, Y.S., Vincent Tseng, S.: A Fast Algorithm for Mining Frequent Closed Itemsets over Stream Sliding Window. In: Proceedings of IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), 996–1002 (2011)
21. Yen, S. J., Wang, C. K., Ouyang, L. Y.: A search space reduced algorithm for mining frequent patterns. Journal of Information Science and Engineering, **28** (1), 177–191 (2012)
22. IBM Synthetic Data Generator http://www.almaden.ibm.com/software/quest/Resorces/index.shtml