iCHUM: An Efficient Algorithm for High Utility Mining in Incremental Databases

Hai-Tao Zheng(✉) and Zhuo Li

Tsinghua-Southampton Web Science Laboratory, Graduate School at Shenzhen, Tsinghua University, Shenzhen, China
zheng.haitao@sz.tsinghua.edu.cn, lizhuo13@mails.tsinghua.edu.cn

Abstract. High utility mining is a fundamental topic in association rule mining, which aims to discover all itemsets with high utility from transaction database. The previous studies are mainly based on fixed databases, which are not applicable for incremental databases. Although incremental high utility pattern (IHUP) mining has been proposed, its tree structure IHUP-Tree is redundant and thus IHUP algorithm has relative low efficiency. To address this issue, we propose an incremental compressed high utility mining algorithm called iCHUM. The iCHUM algorithm utilizes items of high transaction weighted utilization (TWU) to construct its tree structure, namely iCHUM-Tree. The iCHUM algorithm updates iCHUM-Tree when new database is appended to the original database. The information of high utility itemsets is maintained in the iCHUM-Tree such that candidate itemsets can be generated through mining procedure. Performance analysis shows that our algorithm is more efficient than baseline approaches in incremental databases.

Keywords: Data mining · Association rule · High utility mining · Incremental mining

1 Introduction

Mining association rule is a fundamental topic in the data mining applications, especially in market analysis. In association rule mining, frequent pattern mining was firstly proposed to find all itemsets which frequently appear together in the transaction database. The initial solution is based on downward closure property [1,2], which is a level-wise approach. However, it requires multiple database scans and generates a large number of candidate itemsets to search and identity. Extensive studies have been proposed to address the issues by introducing a frequent pattern (FP) tree structure and corresponding FP-growth algorithm [5,6], which is a pattern-growth approach. The main difference between both approaches is whether the database is compressed into other data structure.

However, FP mining is prone to generate many frequent but low profitable itemsets. The reason is that FP mining treats all items with the same weight,

S. Zhang et al. (Eds.): KSEM 2015, LNAI 9403, pp. 212–223, 2015.
DOI: 10.1007/978-3-319-25159-2_20

and each item appears in binary format. In fact, weight and quantity are significant for addressing the decision problems in the real world where sellers require maximizing profit from transaction records [8,9,18,19].

A high utility mining model [13,14] is defined to discover all high utility patterns from the transaction databases. The significance of itemsets is measured by the concept of utility. An itemset is called a high utility itemset if its utility is no less than a user-specified minimum utility threshold represented by *min_util*. Moreover, most previous studies [4,11,13,15,16] are based on a fixed transaction database and have not taken dynamic increase in database size into consideration. In practice, the real markets add their transaction records dynamically, where utility mining for incremental databases is required to be solved.

Studies [3,10,12,17] have been conducted based on incremental databases. IUM and FIUM algorithm [17] are proposed to mine high temporal utility itemsets, which are temporary and may be not high utility itemsets in the whole database. FUP algorithm [10,12] is a level-wise approach, and its efficiency becomes worse due to multiple scans of the whole database. IHUP [3] is a pattern-growth solution for high utility mining in incremental databases. It compresses databases into the IHUP-Tree and avoids multiple scans. IHUP could update its IHUP-Tree when new transaction records are inserted. However, it becomes inefficient when the number of items in the database is relatively large. IHUP maintains its redundant IHUP-Tree, which takes excess time to process items that are unpromising to be high utility itemsets. In fact, not all items need be maintained in tree structure according to transaction weighted downward closure (TWDC) property [13,14].

Most existing methods for fixed databases are not applicable for incremental databases. The approaches for incremental databases spend much time on maintaining redundant information, which causes a relatively low performance. The situations get worse when the size of database becomes large. To address the issues, we propose an incremental compressed high utility mining (iCHUM) algorithm for high utility mining in incremental databases. In this paper, we have three main contributions as below.

1. We propose the iCHUM algorithm for high utility mining in incremental databases. The algorithm performs efficiently to obtain all high utility itemsets as transaction records increase dynamically.
2. The iCHUM algorithm constructs and updates iCHUM-Tree incrementally, which maintains high utility itemsets information of the transaction database. The iCHUM algorithm avoids rebuilding the tree structure entirely and reduces construction and update runtime.
3. We conduct a series of experiments on both real and synthetic datasets. We compare the performance between ours and baseline methods. The results show that our algorithm is more efficient in incremental databases.

The rest of this paper is organized as follows. In Section 2, we introduce the related work. In Section 3, we propose iCHUM algorithm in details following the problem definition. The experimental results and evaluations are shown in Section 4 and the conclusion and future work are given in Section 5.

2 Related Work

The definition of utility mining problem is given in [13,14], which is similar to what we adopt. In their work, they propose the Two-Phase algorithm and introduced the concept of transaction weighted utilization (TWU), which satisfies the downward closure property. The Two-Phase algorithm adopts a level-wise generation-and-test approach. It firstly finds all one-element high TWU itemsets and then generates two-element candidate itemsets to test whether there exist two-element high TWU itemsets or not. If there exist any high TWU candidate itemsets, Two-Phase algorithm generates candidate itemsets by adding one more element. Otherwise, it stops generation process and then identifies utility of all candidate itemsets. It finds all high TWU itemsets level by level, and it needs to scan the whole transaction for each generation-and-test iteration. Therefor, Two-Phase algorithm suffers from the multiple scans of database and huge candidates.

Studies [3,4,11,15] are proposed together with their corresponding tree structure as pattern-growth approaches. The main difference between them is how to construct their tree structure. These approaches scan the database to build their corresponding tree structure separately at first. They compress the entire transaction records into their corresponding tree structure. The mining process utilizes property of prefix-tree and introduces conditional pattern base [6]. Thus the high TWU itemsets are generated by mining the compressed tree structure instead of the whole transaction database. The pattern-growth approaches largely reduce the amount of scans and that of candidates. They perform better in the relatively dense or long pattern databases. However, most pattern-growth methods above are designed for the fixed databases. Their tree structures need to be reconstructed once the database is updated.

There exist some researches based on high utility itemset mining for incremental database [3,10,12,17]. Yeh et al. [17] proposed two methods: incremental utility mining (IUM) algorithm and fast incremental utility mining (FIUM) algorithm. However, these algorithms find high temporal utility itemsets, which are high utility itemsets in the part of the database. When parts joint into the whole, some of high temporal utility itemsets may not be high utility ones. Lin et al. [10,12] proposed an incremental updated HUP maintenance algorithm. The algorithm divides the incremental databases into four cases when new transactions are appended to an original database. However, their algorithm suffers from multiple scans and excessive candidates. Ahmed [3] proposed their incremental mining method IHUP based on the IHUP-Tree. The IHUP-Tree maintains all of the items, and it inserts new transaction records without rebuilding the whole tree. All it needs is to maintain the order of the items by TWU and it is convenient for mining procedure. Limited to the property of their tree structure, the efficiency of the algorithm is not satisfactory when the number of items is large or number of high TWU items is small. On those conditions, IHUP should maintain unnecessary inserting or reordering operation for those low utility items.

Our proposed iCHUM algorithm aims to improve time efficiency by maintaining promising items which may be elements of high utility itemsets.

When transaction database grows incrementally, we recall such high TWU items in new database, whose TWU is low in original database. Then we insert high TWU items of both original and new databases to update the iCHUM-Tree without rebuilding the tree structure entirely. It is efficient to mine the updated iCHUM-Tree to obtain high utility itemsets compared with baseline methods.

3 iCHUM Algorithm

3.1 Problem Definition

Let $I = \{i_1, i_2, \ldots, i_m\}$ be a finite item set. Each item has its profit $p(i_j)$ where $1 \leq j \leq m$. A transaction database consists of a finite set of transaction records $D = \{T_1, T_2, \ldots, T_n\}$ and item profit table. Each transaction $T_s = \{i_{s1}, i_{s2}, \ldots, i_{st}\} \subseteq I$ where $1 \leq s \leq n, 1 \leq t \leq m$. In each T_s, each item has its quantity $q(i_j, T_s)$ where $1 \leq j \leq m, 1 \leq s \leq n$. Let Table 1 be an example of transaction database and Table 2 be a profit table.

Set $X = \{i_{j1}, i_{j2}, \ldots, i_{jl}\} \subseteq I$ is an itemset, where $1 \leq l \leq m$, and l is the length of itemset X. $\forall i_j \in X$, if $i_j \in T_s$, then the itemset $X \subseteq T_s$, which means that T_s contains X.

Table 1. Transaction database

	Tid	Transaction	TU
$D0$	T_1	(A,5) (B,2)	9
	T_2	(A,2) (B,1) (D,1)	6
	T_3	(E,1) (F,2)	7
	T_4	(C,3) (D,2)	13
	T_5	(A,2) (B,2) (C,1) (D,2)	11
$D1$	T_6	(A,3) (B,1) (E,2)	15
	T_7	(B,1) (D,1) (E,1) (F,1)	10
	T_8	(A,2)	2

Table 2. Profit table

Item	Profit
A	1
B	2
C	3
D	2
E	5
F	1

Table 3. TWU table

Item	TWU	
	$D0$	$D0 + D1$
A	26	43
B	26	51
C	24	24
D	30	40
E	7	32
F	7	17

Definition 1. *The utility of an itemset X is denoted as $U(X)$. An itemset X is a **high utility itemset** if $U(X) \geq min\_util$. **High utility mining** is to find the set of all itemsets $\mathbf{X} = \{X_1, X_2, \ldots, X_m\}$ satisfies the condition that $\forall X_i \in \mathbf{X}, U(X_i) \geq min\_util$.*

$$U(X) = \sum_{X \subseteq T_d \in D} \sum_{i_j \in X} p(i_j) \times q(i_j, T_d) \tag{1}$$

After we define utility mining problem, we introduce TWU [13] which helps build up iCHUM-Tree.

Definition 2. *The transaction weighted utility (TWU) of an itemset X denoted as $TWU(X)$ is the sum of the transaction utilities (TU) of all transaction records* *containing X, shown in Table 3. An itemset X is a* ***high TWU itemset*** *if $TWU(X) \geq min\_util$.*

$$TWU(X) = \sum_{X \subseteq T_d \in D} TU(T_d) = \sum_{X \subseteq T_d \in D} \sum_{i_j \in T_d} p(i_j) \times q(i_j, T_d) \tag{2}$$

Noted that $U(X) \leq TWU(X)$ for $\forall X$, if $TWU(X) < min\_util$, then $U(X) < min\_util$. If X is not a high TWU itemset, then X is not a high utility itemset. Moreover, $TWU(X)$ satisfies downward closure property [13,14] while $U(X)$ does not. Therefore, high utility mining problem is divided into high TWU itemsets mining and corresponding utility identification, which constitutes the framework of our algorithm.

Definition 3. *An item $i \in I$ is a promising item if $TWU(i) \geq min\_util$. Otherwise, the item is an unpromising item.*

If an item i_u is unpromising, all itemsets containing i_u should not be high TWU itemsets, which are not high utility itemsets accordingly.

3.2 iCHUM Algorithm Framework

The framework of iCHUM algorithm includes four procedures: iCHUM-Tree construction, iCHUM-Tree update, mining procedure and candidates identifying, which are shown in Fig. 1. The overall inputs consist of the transaction database and a user-specified minimum utility threshold called *min_util*. Actually, for the incremental databases, we have two parts of the transaction database, an original database $D0$ and a new database $D1$. The final output is the collection of high utility itemsets in both database $D0$ and database $D0 + D1$.

According to the flow of the iCHUM framework, we firstly construct an initial iCHUM-Tree from the original database $D0$. Through mining procedure of the iCHUM-Tree, we get the collection of high TWU itemsets in $D0$. We obtain the collection of high utility itemsets in $D0$ following identifying procedure. Then when a new database $D1$ is appended to $D0$ as an incremental database, the iCHUM-Tree is to be updated instead of being rebuilt. The iCHUM-Tree update procedure is to update the iCHUM-Tree according to transaction records in database $D1$. The updated iCHUM-Tree is then the input of mining procedure, which produces the candidate itemsets with high TWU value. The final step of identifying is to pick up real high utility itemsets from the candidates. After that, we obtain all high utility itemsets in the whole transaction database namely $D0 + D1$, which consists of $D0$ and $D1$.

3.3 iCHUM-Tree Construction and Update

The iCHUM-Tree consists of tree structure and its headtable H for traversal, shown in Fig. 2. In the iCHUM-Tree, the nodes includes its name, TWU value, count, a parent node, a brother node and a collection of child nodes, expressed as

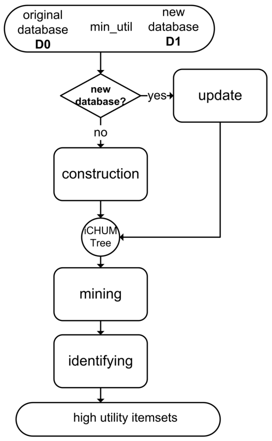
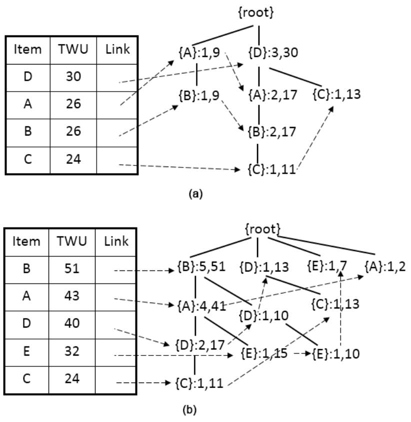


Fig. 1. Framework of iCHUM



Item	TWU	Link
D	30	
A	26	
B	26	
C	24	

Item	TWU	Link
B	51	
A	43	
D	40	
E	32	
C	24	

Fig. 2. Construction and update of iCHUM-Tree (a) after inserting T_5, (b) after inserting T_8

$\{name\}:(count, TWU)$. In the headtable, each entry consists of the item name, TWU value and a link pointed to nodes with the same name in the iCHUM-Tree.

The **iCHUM-Tree construction** is to build iCHUM-Tree from the original database $D0$, shown in Algorithm 1. According to the TWDC property, the items with low TWU value cannot appear in high utility itemsets. Therefore, the headtable of iCHUM-Tree exclusively maintains the promising items whose TWU $\geq min\_util$. Before each transaction record is inserted to iCHUM-Tree, we arrange the items in TWU descending order. It is efficient for mining procedure when traversing branches from bottom to top orderly. When mining process enters entry of higher TWU items, it would not check those low TWU items. If an item has been existed during insertion operation, we add its count by one and its TWU by the TU of current transaction record inserted. Otherwise, we create a node of the item and set its count as one and its TWU as TU value of the current record. The space complexity to construct iCHUM-Tree is $O(m_p^2)$, where m_p represents number of promising items.

Algorithm 1. iCHUM-Tree Construction

```
Input: original database $D0$ , minimum utility threshold $min\_util$
Output: item TWU table $TWU[1..m]$ , iCHUM-Tree and its headtable $H$ of $D0$
  Scan $D0$ to update $TWU[1..m]$
  Create $H$ for each $i$ satisfying $TWU[i] \geq min\_util$ in TWU descending order
  /* Scan and Insert process is as below */
  for each $T_d$ in $D0$ do
    Sort $T_d$ to $T'_d$ in TWU descending order
    Insert i to iCHUM-Tree for each i $\in H$ in $T'_d$
  end for
```

The **iCHUM-Tree update** is performed when a new transaction database $D1$ comes to be appended based on original database $D0$, shown in Algorithm 2. In new database, there would exist such items whose TWU value is high in new database but low in original database. Among such items, **recalled items** are ones whose TWU value is greater than *min_util* in the whole transaction database $D0 + D1$, such as item E in Fig. 2b. In this case, the iCHUM algorithm needs to find the recalled items from $D0$, and insert them to headtable H as well as iCHUM-Tree. Moreover, the iCHUM-Tree should be maintained in TWU descending order after recalled items are appended. We adopt bubble sort operation [7] to reorder the nodes in the iCHUM-Tree and its headtable. The operation exchanges adjacent items to meet the order. If an item X is adjacent to an item Y in headtable and X is Y's parent node in iCHUM-Tree, the bubble sort operation is performed when Y's TWU becomes greater than that of X. The time complexity of update is $O(nm_p^2)$, where n represents number of transaction records. In worst case, it needs to bubble sort n times for m_p entries.

Algorithm 2. iCHUM-Tree Update

```
Input: $D0$ , new database $D1$ , $min\_util$ , $TWU[1..m]$, iCHUM-Tree and its $H$ of $D0$
Output: updated iCHUM-Tree and its headtable $H$ of $D0 + D1$
  Scan $D1$ to update $TWU[1..m]$ and Find collection of recalled items $I'$
  if $I' \neq \emptyset$ then
    Add all $i' \in I'$ to $H$
    Scan $D0$ and Insert $i'$ to iCHUM-Tree
  end if
  Scan $D1$ and Insert $i \in H$ to iCHUM-Tree
  Reorder $H$ and iCHUM-Tree by bubble sort operation
```

Let us give an example of the construction and update procedure. Considering $D0$ in Table 1, we set the *min_util* 40% of the sum of all TU, which is 18.4. The headtable H is created with items whose TWU $\geq$ 18.4. The items of H are "D A B C" in TWU descending order. We insert these items in each transaction by the order and formulate iCHUM-Tree in Fig. 2a. When $D1$ comes, TWU of items changes and *min_util* is 29.2, shown in Table 3. We insert E to the headtable and iCHUM-tree following the previous order. Here, we keep the unpromising item C because it had once been high TWU items. It is likely that it becomes high TWU item again in incremental databases. We rearrange the iCHUM-Tree and its headtable in the "B A D E C" order by bubble sort operation. Not only items in headtable should be sorted in this order, but also the corresponding nodes in the iCHUM-Tree do the same as well. After all items have been sorted, the iCHUM-Tree is updated as shown in Fig. 2b.

3.4 Mining and Identifying Procedures

Mining procedure discovers the collection of high TWU candidate itemsets represented by *cand* from iCHUM-Tree. The mining procedure is a pattern-growth approach, which is shown in Algorithm 3. It is based on the FP-growth mining algorithm [6]. We firstly construct conditional tree CT_α for each entry α in H. We follow the link in α's entry to obtain all transaction records containing item α. For each α node in iCHUM-Tree, we find its prefix nodes and calculate TWU of its prefix nodes based on TWU of α. If prefix node β's TWU $\geq$ *min_util* in CT_α, then we add β to CT_α and add $\{\alpha\beta\}$ to *cand*. The time complexity to mine iCHUM-Tree is $O(hm_p^2)$, where h is the height of iCHUM-Tree and m_p represents number of promising items. The mining recursion complexity is determined by the height and node number of iCHUM-Tree.

For example, we construct B's conditional tree CT_B from iCHUM-Tree in Fig. 2a. For node {B}:(1,7), we obtain its prefix node A as {A}:(1,9). In terms of node {B}:(2,17), we add A's count and TWU by 2 and 17 respectively. Besides, we obtain B's another prefix node D as {D}:(2,17). The prefix nodes of B is now that {A}:(3,26) and {D}:(2,17). Considering *min_util* is 18.4, B's conditional tree consists of node A and discards node D. We add itemset {AB} into set *cand*. For {AB}'s prefix nodes, we can obtain that {D}:(2,17), which is less than *min_util*. Mining procedure for B is finished and then iCHUM continues to process item A. After iCHUM processes each entry in headtable H, we can obtain the set *cand* containing {C},{CD},{B},{AB},{A},{D} in original database $D0$.

Algorithm 3. Mining procedure

```
Input: $min\_util$ , iCHUM-Tree, headtable $H$
Output: collection of high TWU itemsets $cand$
  for each entry $\alpha$ in $H$ do
    Add $\{\alpha\}$ to $cand$
    Call Mining($\mathrm{CT}_\alpha$, $H_\alpha$, $\alpha$)
  end for
  proc Mining($\mathrm{CT}_\alpha$,$\mathrm{H}_\alpha$, $\alpha$)
    Create $\alpha'$s conditional tree $\mathrm{CT}_\alpha$ from iCHUM-Tree
    Create headtable $H_\alpha$ for $\mathrm{CT}_\alpha$
    for each entry $\beta$ in $H_\alpha$
      Add $\{\beta\} \cup \{\alpha\}$ to $cand$
      Call Mining($\mathrm{CT}_{\alpha\beta}$, $H_{\alpha\beta}$, $\alpha\beta$)
    end for
```

Identifying procedure is to calculate the utility of these itemsets in the collection of high TWU candidate itemsets *cand* according to Definition 1. The iCHUM algorithm needs to rescan the transaction database to obtain utility of all itemsets in *cand*. For those recalled items and their corresponding itemsets, we need to rescan the original database as well as the new database. We calculate their utility in each transaction records and sum them up. For other items, we have had calculated their utility in the identifying process of $D0$. We only rescan the new database for those itemsets and add to their utility.

In the above example, we calculate the utility of each itemset in *cand* according to Table 1. $U(\{C\}), U(\{B\}), U(\{A\}), U(\{D\})$ is 12, 10, 9, 10 respectively.

The utilities of items in headtable H are less than min_util. For two-element itemsets, $U(\{CD\})$ is 20 and $U(\{AB\})$ is 19, which are high utility itemsets for transaction database $D0$. After a new database $D1$ is inserted, the updated iCHUM-Tree is as shown in Fig. 2b. Following mining procedure, we obtain the *cand* of the database $D0+D1$, which includes {E},{D},{A},{AB},{B}. For {E}, we rescan the whole transaction database and obtain its utility as 20. For the rest, we add their utility in $D1$ and obtain their final utility as 12, 14, 24, 14. Because the utility of each candidate itemset is less than min_util 29.2, there is no high utility itemset for transaction database $D0 + D1$ when min_util is set as 40% of total transaction utilities.

4 Experiment

In this section, we evaluate the performance of iCHUM algorithm written in C++. The experiment were conducted on Ubuntu server with a dual-2.4GHz CPU processor and 4G memory. Both real and synthetic dataset could be obtained from NU-MineBench [1]. Real dataset, named *Chainstore*, is a sparse and large database. It contains 1,112,949 transaction records and total 46,086 kinds of items. We split the database into $D0$ of 700,000 and $D1$ of 412,949. Synthetic dataset, named T10I6D100, contains 100 items and 93,058 transaction records whose average length is 10, where $|D0|$ is 60,000 and $|D1|$ is 33,058.

As a comparison, we implement the IHUP algorithm [3] and an iCHUM without update (iCHUMxU) algorithm in C++ as baseline methods. The iCHUMxU algorithm mines iCHUM-Tree twice without update procedure regarding the $D0$ and $D0 + D1$ as original database input respectively. The iCHUMxU is a high utility mining algorithm for fixed database and we compare its mining efficiency with iCHUM's in incremental databases.

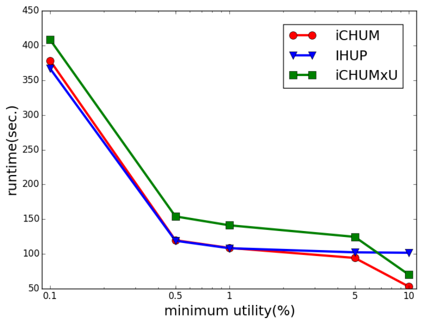


Fig. 3. Runtime on T10I6D100

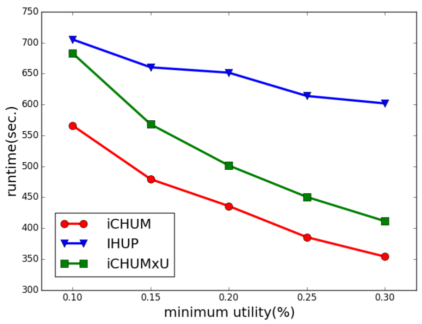


Fig. 4. Runtime on Chainstore

Fig. 3 shows the execution runtime on T10I6D100. With a logarithmical X axis, the runtime at different minimum utility thresholds is easy to view.

[1] http://cucis.ece.northwestern.edu/projects/DMS/MineBench.html

For larger *min_util* or minimum utility, runtime of iCHUM is less than that of IHUP. When *min_util* is 10%, the total runtime of iCHUM is 52.99 seconds, compared with 101.65 seconds of IHUP and 70.17 seconds of iCHUMxU. It is because that the items in iCHUM-Tree is a small part of the total items. It takes less time on update and does not need to reconstruct the whole iCHUM-Tree. However, the performance of iCHUM algorithm becomes worse when *min_util* gets smaller compared with IHUP. When *min_util* is 0.1%, the total runtime of iCHUM is 378.43 seconds, which is larger than 366.97 seconds of IHUP. It is because that the number of items with high TWU value reaches close to the total items number. In time complexity, m_p is approximated with number of items m. The cost of maintaining headtable is almost the same. Besides, the iCHUM algorithm should spend more time on finding back recalled items. That happens in such a dataset, where there are less items and transaction length is longer.

The iCHUM algorithm has an advantage over IHUP on sparse and large database, shown in Fig. 4. In Chainstore dataset, each item accounts for small proportion of whole transaction ($m_p \ll m$), where maintaining headtable with high TWU items is efficient. When *min_util* is 0.2%, total runtime of iCHUM is 435.86 seconds, which is less than 651.61 seconds of IHUP. IHUP should maintain entire items and takes overhead time, which is larger than 501.41 seconds of iCHUMxU. The reason is that valuable items in a large transaction database is rare, and thus it is efficient to maintain these promising items instead of entire items. From runtime distribution in Table 4, runtime is largely reduced in construction and update procedures, which the iCHUM algorithm focuses on. The fewer items save the execution time on construction and update of the iCHUM-Tree. Besides, the iCHUMxU is a time-consuming method in both datasets. It is more efficient to update iCHUM-Tree than to rebuild the tree structure entirely since we reuse the previous tree structure and mining results.

Table 4. Runtime Distribution (sec.) of iCHUM and IHUP

Dataset	Algorithm	D0			D1			Time
		construction	mining	identify	update	mining	identify	
T10I6D100	IHUP	33.89	0.27	0.07	67.40	0.42	0.22	102.27
0.5%	iCHUM	**30.60**	0.26	0.07	**62.64**	0.41	0.22	**94.20**
Chainstore	IHUP	209.14	1.45	46.09	318.78	2.29	73.86	651.61
0.2%	iCHUM	**113.98**	1.33	46.00	**199.74**	2.18	72.63	**435.86**

To verify whether the iCHUM algorithm obtains all high utility itemsets, we keep records of mining results from both iCHUM and IHUP. Moreover, we compare the results on different datasets with that of Two-Phase algorithm provided by NU-MineBench [13,14]. Table 5 shows that the number of high utility itemsets at different minimum utility threshold on T10I6D100 dataset. Table 6 shows the mining results on Chainstore dataset.

Table 5. Number of High Utility Itemsets on T10I6D100

Database	Minimum Utility Threshold				
	10%	5%	1%	0.5%	0.1%
$D0$	0	2	292	1643	46471
$D0 + D1$	0	2	292	1644	45614

Table 6. Number of High Utility Itemsets on Chainstore

Database	Minimum Utility Threshold				
	0.30%	0.25%	0.20%	0.15%	0.10%
$D0$	15	18	29	48	86
$D0 + D1$	15	17	26	50	80

5 Conclusion and Future Work

In this paper, we propose an efficient iCHUM algorithm for mining high utility itemsets in incremental databases. The iCHUM algorithm compresses transaction database into a compact tree structure called iCHUM-Tree. The update of iCHUM-Tree maintains the tree structure with all promising items which guarantees all high utility itemsets to be found. Experimental analysis shows that iCHUM performs better than other baselines in incremental databases, especially in terms of those with large number of transaction records or items. We believe that iCHUM algorithm will play an important role for high utility mining in incremental databases in practice.

We notice that the performance of the iCHUM algorithm degrades as the amount of the recalled items increases. In the future, we will explore a knowledge-based method to improve the promising item discovering. In addition, we will study the idea of B+ tree to improve our data structure and algorithm in construction and mining procedures.

Acknowledgments. This research is supported by the 863 project of China (2013AA013300), National Natural Science Foundation of China (Grant No. 61375054 and 61402045), Tsinghua University Initiative Scientific Research Program (Grant No.20131089256), and Cross fund of Graduate School at Shenzhen, Tsinghua University (Grant No. JC20140001).

References

1. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. ACM SIGMOD Record **22**(2), 207–216 (1993)
2. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: Proceedings of the 20th International Conference on Very Large Data Bases, pp. 487–499 (1994)
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. Erwin, A., Gopalan, R.P., Achuthan, N.: Ctu-mine: an efficient high utility itemset mining algorithm using the pattern growth approach. In: 2007 7th IEEE International Conference on Computer and Information Technology, pp. 71–76 (2007)
5. Grahne, G., Zhu, J.: Fast algorithms for frequent itemset mining using fp-trees. IEEE Transactions on Knowledge and Data Engineering **17**(10), 1347–1362 (2005)
6. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. ACM SIGMOD Record **29**(2), 1–12 (2000)
7. Koh, J.-L., Shieh, S.-F.: An efficient approach for maintaining association rules based on adjusting fp-tree structures. In: Lee, Y.J., Whang, K.-Y., Li, J., Lee, D. (eds.) DASFAA 2004. LNCS, vol. 2973, pp. 417–424. Springer, Heidelberg (2004)
8. Li, Y.C., Yeh, J.S., Chang, C.C.: Efficient algorithms for mining share-frequent itemsets. In: Proceedings of the 11th International Fuzzy Systems Association World Congress, pp. 534–539 (2005)
9. Li, Y.C., Yeh, J.S., Chang, C.C.: Isolated items discarding strategy for discovering high utility itemsets. Data & Knowledge Engineering **64**(1), 198–217 (2008)
10. Lin, C.W., Hong, T.P., Lu, W.H.: Maintaining high utility pattern trees in dynamic databases. In: 2010 2nd International Conference on Computer Engineering and Applications, pp. 304–308 (2010)
11. Lin, C.W., Hong, T.P., Lu, W.H.: An effective tree structure for mining high utility itemsets. Expert Systems with Applications **38**(6), 7419–7424 (2011)
12. Lin, C.W., Lan, G.C., Hong, T.P.: An incremental mining algorithm for high utility itemsets. Expert Systems with Applications **39**(8), 7173–7180 (2012)
13. Liu, Y., Liao, W.K., Choudhary, A.: A fast high utility itemsets mining algorithm. In: Proceedings of the 1st International Workshop on Utility-based Data Mining, pp. 90–99 (2005)
14. Liu, Y., Liao, W., Choudhary, A.K.: A two-phase algorithm for fast discovery of high utility itemsets. In: Cheung, D., Ho, T.-B., Liu, H. (eds.) PAKDD 2005. LNCS (LNAI), vol. 3518, pp. 689–695. Springer, Heidelberg (2005)
15. Tseng, V.S., Wu, C.W., Shie, B.E., Yu, P.S.: Up-growth: an efficient algorithm for high utility itemset mining. In: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 253–262 (2010)
16. Wu, C.W., Shie, B.E., Tseng, V.S., Yu, P.S.: Mining top-k high utility itemsets. In: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 78–86 (2012)
17. Yeh, J.S., Chang, C.Y., Wang, Y.T.: Efficient algorithms for incremental utility mining. In: Proceedings of the 2nd International Conference on Ubiquitous Information Management and Communication, pp. 212–217 (2008)
18. Yun, U.: Efficient mining of weighted interesting patterns with a strong weight and/or support affinity. Information Sciences **177**(17), 3477–3499 (2007)
19. Yun, U., Leggett, J.J.: Wfim: weighted frequent itemset mining with a weight range and a minimum weight. In: Proceedings of the 2005 SIAM International Conference on Data Mining, pp. 636–640 (2005)