

# A Bottom-Up Projection Based Algorithm for Mining High Utility Itemsets

Alva Erwin<sup>1</sup>, Raj P. Gopalan<sup>1</sup>, N.R. Achuthan<sup>2</sup>

<sup>1</sup>Department of Computing, <sup>2</sup>Department of Mathematics and Statistics  
Curtin University of Technology, Kent St. Bentley Western Australia  
alva.erwin@postgrad.curtin.edu.au, r.gopalan@curtin.edu.au, n.r.achuthan@curtin.edu.au

## Abstract

*Mining High Utility Itemsets from a transaction database is to find itemsets that have utility above a user-specified threshold. This problem is an extension of Frequent Itemset Mining, which discovers itemsets that occur frequently (i.e. with occurrence count larger than a user given value). The problem of finding High Utility Itemsets is challenging, because the anti-monotone property so useful for pruning the search space in conventional Frequent Itemset Mining does not apply to it. In this paper we propose a new algorithm called CTU-PRO that mines high utility itemsets by bottom up traversal of a compressed utility pattern (CUP) tree. We have tested our algorithm on several sparse and dense data sets, comparing it with the recent algorithms for High Utility Itemset Mining and the results show that our algorithm works more efficiently.*

**Keywords:** High Utility Itemset Mining, Pattern Growth.

## 1 Introduction

Mining Frequent Itemsets from transaction databases is a fundamental task for several forms of knowledge discovery such as association rules (R. Agrawal, Imielinski, & Swami, 1993), sequential patterns (Rakesh Agrawal & Srikant, 1995; Han et al., 2000) and classification (B. Liu, Hsu, & Ma, 1998; Yudho Giri Sucahyo & Raj P. Gopalan, 2004). Since its introduction in (R. Agrawal, Imielinski, & Swami, 1993), a number of Frequent Itemset Mining algorithms have been proposed (Han, Wang, & Yin, 2000; Y.G Sucahyo & R.P Gopalan, 2004; M.J. Zaki, 2000). The goal of Frequent Itemset Mining is to find items that co-occur above a user given value of frequency, in the transaction database. In the Frequent Itemset Mining problem, the occurrence of each item in a transaction is represented by a binary value without considering its quantity or an associated weight such as price or profit. However, quantity and weight are significant for addressing real world decision problems that require maximizing the utility in an organization. For example, selling a laser printer may occur less frequently than sale of printer ink in an electronic superstore, but the former gives a much higher profit per unit sold. The high utility itemset mining problem is to find all itemsets that

have utility larger than a user specified value of minimum utility.

The Frequent Itemset Mining algorithms use the Apriori principle (R. Agrawal, Imielinski, & Swami, 1993) that any superset of a non-frequent itemset is also non-frequent. This *anti-monotone* property of itemsets is used to reduce the search space by pruning the non-frequent itemsets early. The Apriori principle does not hold for itemset utility, since the superset of a low utility itemset may not be a low utility itemset. For example, if itemset {*printer ink*} has a low utility, its superset, {*printer ink, color laser printer*} might have a high utility, and so we cannot prune {*printer ink*}.

A framework for high utility itemset mining was proposed recently by Yao et al (Hong Yao & Hamilton, 2006; H. Yao, Hamilton, & Buzz, 2004). They proposed a mining method and described pruning strategies based on the mathematical properties of utility constraints. They also developed an algorithm named *Umining* and another heuristic based algorithm *Umining\_H* to discover high utility itemsets.

Recent research has focused on efficient high utility mining using intermediate anti-monotone measures for pruning the search space. Liu et al. (Y. Liu, Liao, & Choudhary, 2005) propose a two phase algorithm to mine high utility itemsets. They use a transaction weighted utility (TWU) measure in the first phase to find supersets of high utility itemsets, followed by a rescan of the database to determine the actual high utility itemsets among them. However, their algorithm is based on the candidate generation-and-test approach and so suffers from poor performance when mining dense datasets and long patterns much like the Apriori algorithm for frequent pattern mining.

In this paper, we propose an efficient algorithm for utility mining using the pattern growth approach (Han, Wang, & Yin, 2000). Our recent research on utility mining (Erwin, Gopalan, & Achuthan, 2007) had produced an algorithm that ran efficiently on dense data, but performed unsatisfactorily on sparse data. Motivated by the need to improve the performance on sparse data, we have developed a new compact data representation named *Compressed Utility Pattern tree (CUP-tree)* which extends the *CFP-tree* of (Y.G Sucahyo & R.P Gopalan, 2004) for utility mining, and a new algorithm named *CTU-PRO* for mining the complete set of high utility itemsets. The concept of TWU is used for pruning the search space in *CTU-PRO*, but it avoids a rescan of the database to determine the actual utility of high TWU itemsets. The algorithm creates a *CUP-Tree* named *GlobalCUP-Tree* from the transaction database after first identifying the individual high TWU items. For each high

TWU item, a smaller projection tree called *LocalCUP-Tree* is extracted from the *GlobalCUP-tree* for mining all high utility patterns beginning with that item as prefix. The performance of *CTU-PRO* is compared against the implementation of the TwoPhase algorithm (Y. Liu, Liao, & Choudhary, 2005) available from (CUCIS) and also with *CTU-Mine* (Erwin, Gopalan, & Achuthan, 2007). The experimental results show that *CTU-PRO* performs better than the previous algorithms on both sparse and dense datasets at most support levels.

The rest of the paper is organized as follows: In Section 2, we define the relevant terms. In Section 3, we present the *CUP-tree* data structure. Section 4 describes the *CTU-PRO* algorithm. The performance studies of the algorithm are given in section 5. Section 6 is the conclusion of the paper and includes pointers for possible future research.

## 2 High Utility Itemset Mining

In this Section, we give the basic notations and the definitions of terms to describe high utility itemset mining, based on (H. Yao, Hamilton, & Buzz, 2004; Hong Yao, Hamilton, & Geng, 2006). Let  $I = \{i_1, i_2, \dots, i_m\}$  be a set of items and  $D = \{T_1, T_2, \dots, T_n\}$  be a transaction database where the items of each transaction  $T_i$  is a subset of  $I$ . The quantity of an item  $i_p$  in a transaction  $T_q$  is denoted by  $o(i_p, T_q)$ . The external utility  $s(i_p)$  is the value of a unit of item  $i_p$  in the utility table, (e.g., profit per unit). The utility of item  $i_p$  in transaction  $T_q$ , denoted by  $u(i_p, T_q)$  is defined as  $o(i_p, T_q) \times s(i_p)$ . A set  $X$  is called an itemset if  $X$  is a subset of  $I$ . The utility of  $X$  in transaction  $T_q$ , denoted by  $u(X, T_q)$  is defined as :

$$u(X, T_q) = \sum_{i_p \in X} u(i_p, T_q). \quad (1)$$

The utility of itemset  $X$  in the database, denoted by  $u(X)$  is defined as:

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

The task of high utility itemset mining is to find all itemsets that have utility above a user-specified  $min\_utility$ . Since utility is not *anti-monotone*, Liu et al. (Y. Liu, Liao, & Choudhary, 2005) proposed the concepts of Transaction Utility (TU) and Transaction Weighted Utility (TWU) to prune the search space of high utility itemsets. Transaction Utility of a transaction, denoted  $tu(T_q)$  is the sum of the utilities of all items in  $T_q$ :

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

*Transaction Weighted Utility* of an itemset  $X$ , denoted as  $twu(X)$  is the sum of the transaction utilities of all the transactions containing  $X$ :

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

An item  $i \in I$ , is said to be an *individual high utility item* if

$$twu(\{i\}) = \sum_{T_q \in D \wedge \{i\} \subseteq T_q} tu(T_q) \geq min\_utility. \quad (5)$$

As shown in (Y. Liu, Liao, & Choudhary, 2005), any superset of a low TWU itemset is also a low TWU itemset. Thus we can use TWU to prune the supersets of low TWU itemsets. However, since TWU is an over-estimation of the real utility itemset value, further pruning of high TWU itemsets will be required for mining high utility itemsets.

Suppose we have a small transaction database of an electronic retailer as shown in Figure 1(a). Figure 1(b) shows the profit (external utility) for each item. The values in each row in figure 1(a) shows the quantity of each item bought in a particular transaction, (i.e. the local transaction utility value). The last column shows the transaction utility for each transaction with total transaction utility of the database in the last row. In transaction  $t_1$ , two A (printer ink), one C (bubble jet printer), and one D (digital camera) were bought, yielding transaction utility of \$80. The utility of item A,  $u(A, t_1)$  is \$20 and the utility of item A in the whole database,  $u(A) = \$120$ . Itemset  $CD$  occurs 2 times, in transactions  $t_1$  and  $t_3$ .

TID	A	B	C	D	E	F	Transaction Utility (\$)
$t_1$	2	0	1	1	0	0	80
$t_2$	2	1	1	0	0	0	195
$t_3$	0	0	1	1	10	0	110
$t_4$	0	1	0	0	15	0	225
$t_5$	1	0	1	0	0	1	37
$t_6$	2	0	0	1	10	0	105
$t_7$	2	0	0	0	8	1	62
$t_8$	1	1	0	1	2	0	205
$t_9$	1	0	0	1	10	0	95
$t_{10}$	1	1	0	0	5	0	185
<i>total</i>	12	4	4	5	60	2	1299

(a) Transaction database

		Profit (\$)
A	Printer Ink	10
B	Colour Laser Printer	150
C	Bubble Jet Printer	25
D	Digital Camera	35
E	Glossy Photo Paper	5
F	Floppy Disk	2

(b) Utility table

Figure 1. An example transaction database and utility table

Further,  $u(CD, t_1) = \$60$ ,  $u(CD, t_3) = \$60$  and  $u(CD) = \$120$  and  $twu(CD) = \$190$ . The TWU of item B (colour laser printer) is the sum of the transaction utilities of ( $t_2, t_4, t_8, t_{10}$ ) = \$810 and the TWU of itemset AC is the sum of transaction utilities of ( $t_1, t_2, t_3$ ) = \$312.

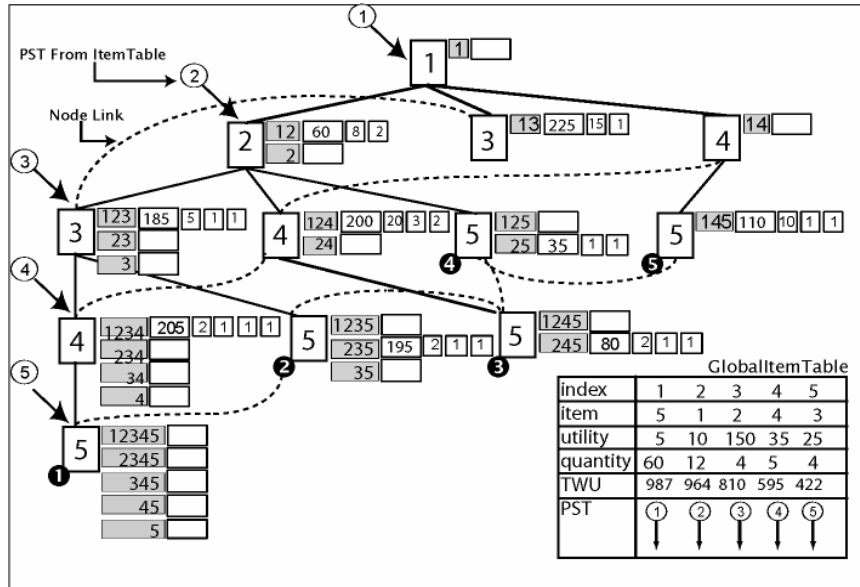


Figure 2. *GlobalCUP-Tree* and *GlobalItemTable*

### 3 CTU-PRO Data Structure

In this section, a new tree-based data structure, named *Compressed Utility Pattern-Tree (CUP-Tree)*, is introduced. It is an extension of *CFP-Tree* (Y.G. Sucahyo & R.P. Gopalan, 2004) and a variant of *CTU-Tree* (Erwin, Gopalan, & Achuthan, 2007). In *CUP-tree*, the itemset counts of *CFP-tree* are replaced by the corresponding TWUs and arrays of total quantities of items that make up the itemsets. *CUP-Tree* is a variant of *CTU-Tree* with a pointer link connecting nodes containing the same item identifier. This feature of *CUP-tree* enables the bottom-up traversal of the tree while mining.

**Definition 1 (Compressed Utility Pattern-Tree or CUP-Tree).** A *CUP-Tree* is a prefix tree with the following properties:

1. It consists of an *ItemTable* called *GlobalItemTable* made up of all high TWU items and a tree called *GlobalCUP-Tree* with nodes linked by pointers from *GlobalItemTable*, containing pattern information for utility mining as described below.
2. *GlobalItemTable* contains all individual high TWU (hTWU) items sorted in descending order by their TWU values. Each entry in *GlobalItemTable* consists six fields: (1) index, (2) item-id, (3) utility per unit of the item, (4) Total quantity of the item, (5) TWU of the item, and (6) a pointer to the root of the subtree of the item in *GlobalCUP-Tree*.
3. The root of *GlobalCUP-Tree* which is at level 0 of the tree represents the index of the item with the highest TWU. The corresponding item entry in *GlobalItemTable* contains a pointer to this node.
4. Each node in *GlobalCUP-Tree* consists of five fields: (1) node-id, (2) a pointer to the next sibling, (3) a link pointer to a node (at an immediate predecessor level of the tree) with the same item id, (4) a TWU array where each entry corresponds to the transaction utility of the itemset, and (5) an array of pointers to the quantities of items in the itemset.

5. If  $T = \{T_0, T_1, \dots, T_k\}$  is a set of TWU values in the TWU array attached to a node where the index of the array starts from zero, then  $T_i$  is the TWU of a transaction with an itemset along the path from the node at level  $i$  to the node where  $T_i$  is located.
6. If  $I = \{i_1, i_2, \dots, i_k\}$  is a set of hTWU items in a transaction, after being mapped to their index-id, then the transaction will be inserted into the *CUP-Tree* starting from the root of a subtree pointed from the entry of item  $i_1$  in *ItemTable*.

Based on the above description, a *GlobalCUP-Tree* can be constructed as follows:

**Step 1. Construct *GlobalItemTable*.** Initially the table contains just the row headings and none of the items. The database is scanned once. On reading each transaction, the *ItemTable* is updated as follows:

- (a) For each item present in the transaction and not yet recorded in the *ItemTable*, generate a new column of the table and initialize the entries, namely, the original item-id, utility per unit of the item, total quantity of the item in the transaction, and the total utility of this transaction. Note that the mapped item-id and the pointer to the root of the subtree will be null at this stage.
- (b) For each item present in the transaction and in the *ItemTable*, update the following entries in the column corresponding to the item: add the total quantity of the item in the transaction to the existing entry; add the total utility of this transaction to the existing entry.

On completing the first scan of the database, the *GlobalItemTable* is reorganized as follows:

- (a) Delete the items and the respective columns from the table if TWU of the item is less than *min\_utility*.
- (b) Sort the remaining items of the table termed as individual high TWU (1-hTWU) items in ascending order of their TWU values.
- (c) The sorted item-ids are mapped to new identifiers that are integers in ascending sequence.

- (d) For each item-id in ItemTable, associate a pointer to the root of the subtree of the corresponding mapped new item-id in *CUP-Tree*.

Consider the transaction database and the utility table of items given in Figure 1. Suppose the user wants to mine the high utility itemsets with a minimum utility of 10% of the total transaction utility in the database, which is equal to \$129.9. On completing the database scan, the items with TWU less than  $min\_utility$  will be pruned. For our example, the item 6 is excluded, since its TWU of 99 is less than  $min\_utility$ . The constructed *GlobalItemTable* is shown in Figure 2. (We refer to the main ItemTable as *GlobalItemTable* to distinguish it from *LocalItemTable* of the projection tree used in the mining process to be described in Section 4). The mapped new item-ids of 1 to 5 correspond to the original items 5, 1, 2, 4 and 3 respectively.

**Step 2. Construct Global Compressed Utility Pattern Tree (CUP-Tree).** First initialize *GlobalCUP-Tree* with root node identified by mapped item-id 1. For each item  $i$  in *GlobalItemTable*, attach a pointer to the root of the subtree generated by the node  $i$  of the initial *CUP-Tree*. This pointer will be used as a starting point to mine all high utility itemsets using *CUP-Tree*. Next, the database is scanned once to complete the construction of *GlobalCUP-Tree*. On reading a transaction, we insert into *GlobalCUP-Tree* the relevant patterns and related information for the 1-hTWU items listed in *GlobalItemTable*. More specifically, the *CUP-Tree* is updated as follows after scanning each transaction: Let  $i_1, i_2, \dots, i_k$  be the 1-hTWU items in the transaction in increasing order of the mapped item-ids. If a pattern generated by this transaction is not present in the current *CUP-Tree*, a node is created as part of an appropriate subtree and all the relevant patterns are recorded along with the updated TWU and the array of total quantities.

Original item id	1	2	3	4	5	twu
$t_1$	2	0	1	1	0	80
$t_2$	2	1	1	0	0	195
...	...	...	...	...	...	...
$t_0$	1	1	0	0	5	185

<i>GlobalItemTable</i> index	1	2	3	4	5	twu
$t_1$	0	2	0	1	1	80
$t_2$	0	2	1	0	1	195
...	...	...	...	...	...	...
$t_0$	5	1	1	0	0	185

<i>GlobalItemTable</i> index	1	2	3	4	5
Original item id	5	1	2	4	3

Figure 3. Mapping from original transaction database using *GlobalItemTable*

If a pattern generated by this transaction is present in some node of the current *CUP-Tree*, then we update the TWU and the array of total quantities associated with it. While updating the TWU for any pattern the contributions of the items not included in *GlobalItemTable* are deducted from the corresponding transaction value. A node link is created for each item entry in *GlobalItemTable* linking all nodes of the tree with the same mapped item-id.

Consider the transaction database of Figure 1 and *GlobalItemTable* in Figure 2. Each transaction is inserted into the tree using the mapping in *GlobalItemTable*. For example, transaction 1 has mapped item-ids 2, 4 and 5 (for the original items 1, 4 and 3) with respective quantities 2, 1, and 1 (see Figure 3). This pattern is inserted into the tree with the TWU (80) and a pointer to the array of quantities 2, 1 and 1 (See Figure 3 for the mapping). In Figure 2, the nodes with the same mapped item-ids of 3, 4 and 5 are linked to the corresponding entries in *GlobalItemTable*.

```

1 begin
2 // Step 1: Construct GlobalItemTable
3 for each transaction t ∈ D
4   for each item i ∈ t
5     if i ∈ GlobalItemTable
6       Increment quantity and TWU of i
7     else
8       Insert i into GlobalItemTable
9         with quantity and TWU of i
10    end if
11  end for
12 Sort GlobalItemTable in TWU descending order
13 Assign an index for each hTWU Item in the
14   GlobalItemTable
15 // Step 2: Construct CUP-Tree
16 Construct the left most branch of the tree
17 for each transaction t ∈ D
18   Initialize mappedTrans
19   for each hTWU item i ∈ t
20     mappedTrans = mappedTrans ∪ getIndex(i)
21   end for
22   Sort mappedTrans in ascending order of item
23   ids
24   InsertToCUPtree(mappedTrans)
25 end for
26 end
27 Procedure InsertToCUPtree(mappedTrans)
28   firstItem := mappedTrans[1]
29   currNode := root of subtree pointed
30   by ItemTable[firstItem]
31   for each subsequent item i ∈ mappedTrans
32     if currNode has child represent i
33       Increment quantity and TWU
34       [firstItem-1] of the child node
35     else
36       Create child node and set its
37       quantity and TWU [firstItem-1] to its
38       respective value
39       Link the node to its respective nodelink
40     end if
41   end for
42 end

```

Figure 4. CUP-Tree Data Structures Construction

## 4 CTU-PRO Algorithm

In this Section, a new algorithm that traverses the tree using a bottom-up approach is presented. The main features of the *CTU-PRO* are: (1) the least individual high TWU item, say with index  $j$  of the *GlobalItemTable* is first selected to mine for the high TWU itemsets that include index  $j$ . (2) Using this item index  $j$ , *CTU-PRO* focuses on a projection of all the transactions ending with index  $j$ . (3) This is accomplished by entering the *GlobalCUP-tree* at the pointer provided by item index  $j$  and developing a *LocalCUP-Tree* and a *LocalItemTable* while traversing the *GlobalCUP-Tree* using the node-link (bottom-up). The mining process in *CTU-PRO* is described by the algorithm in Figure 6 and illustrated by the following example. Let the *CUP-Tree*, as shown in Figure 2, be the input for the mining step in *CTU-PRO*.

We construct another tree called a High Utility Pattern Tree (*HUP-Tree*) to record the high utility patterns and their utility values computed by traversing the *LocalCUP-Tree*. Figure 7 shows the *LocalCUP-Tree* and *HUP-Tree* for the projections of mapped indexes 5, 4, 3 and 2 during the mining process. The mining steps are explained below using the example data.

**4.1. Construction of LocalItemTable.** *CTU-PRO* starts from the least hTWU item (index: 5, item: 3) in the *GlobalItemTable* (line 3). It creates a projection of all transactions ending with nodes of index 5. This projection is represented by a *LocalCUP-Tree* and only contains items that are hTWU locally. Traversing the *node-link* of index 5 in the *GlobalCUP-Tree* identifies the local hTWU items that occur together with it. There are five nodes of index 5 identified by ❶..❺ in Figure 2. The path to the root for each node is traversed, computing TWU of the other indexes that occur together with index 5 (lines 16-26). In all, we have 1 (110) (from ❺), 2 (310) (from ❷, ❸, ❹), 3 (195) (from ❷) and 4 (190) (from ❸ and ❺) for index 5. Since our *min\_utility* is 129.9, indexes 2, 3, 4 (item id: 1,2,4) are *locally hTWU*. They are registered in the *LocalItemTable* and assigned new index ids (see Figure 7a). They also become the child nodes of the *HUP-Tree* root (lines 7-9). During the traversal, we also keep the quantity of the item and the quantity of the projection item that co-occurred with that item. As an example, for index 2, when we visit the nodes ❷, ❸, ❹, we will store the cumulative quantity of 2 (=5) and cumulative quantity of index 5 as the current projection (=3) in *LocalItemTable*.

<i>GlobalItemTable</i> index	1	2	3	4	5	twu
❶	0	0	0	0	0	-
❷	0	2	1	0	1	195
❸	0	2	0	1	1	80
❹	0	1	0	0	1	35
❺	10	0	0	1	1	110

<i>GlobalItemTable</i> index	1	2	3	4	5
Original item index	5	1	2	4	3
<i>LocalItemTable</i> index	-	1	2	3	-

Figure 5. Mapping from projection of index 5 using *LocalItemTable*

This makes the *LocalItemTable* slightly different from *GlobalItemTable* which we described previously. (Note that we have an additional field in the *LocalItemTable* in Figure 7). Together, the root and its children form the high TWU itemsets of length two. Then the calculation of real utility for these 2-itemsets can be done.

**4.2. Construction of Local CUP-Tree.** After local hTWU items for the projection have been identified, the *node-link* in the *GlobalCUP-Tree* is re-traversed and the path to the root from each node is revisited to get the local hTWU

items occurring together with index 5 in the transactions. Once again the items are mapped to new local item-ids.

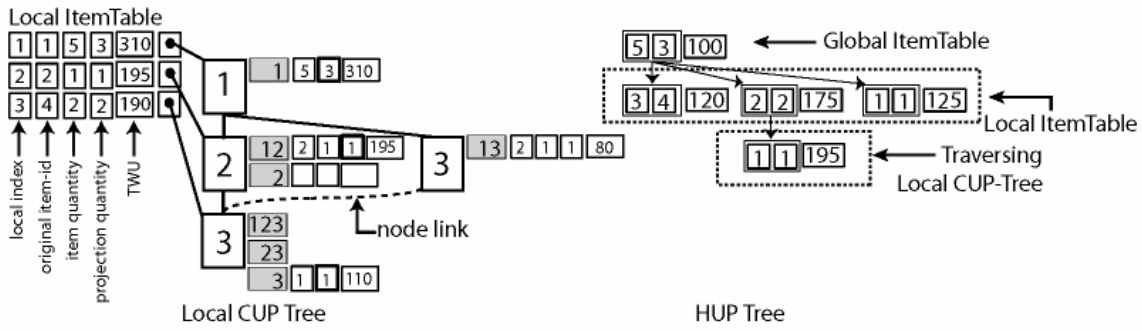
```

input CUP-Tree
output High Utility Itemsets HUI
1 Procedure Mining
2 // Step 1: Construct LocalItemTable
3 for each hTWU item i ∈ GlobalItemTable
4   from the least to the most hTWU
5   Initialize HUPTree with i as the
6   Root of the tree
7   ConstructLocalItemTable(x)
8 // Step 2: Construct LocalCUPTree
9 for each hTWU item j ∈ LocalItemTable
10  Attach j as a child of i
11 end for
12 ConstructLocalCUPTree(x)
13 // Step 3: Mining Database Projection
14 RecMine(x)
15 Traverse the HUPTree to print the
16 high utility itemsets
17 end for
18 end
19 Procedure ConstructLocalItemTable(i)
20 for each occurrence of node i in the
21 GlobalCUP-Tree
22 for each item j in the path to the root
23 if j ∈ LocalItemTable
24   Increment quantity and TWU of j
25 else
26   Insert j into LocalItemTable
27   along with quantity of j and
28   quantity of i
29 end if
30 end for
31 end for
32 Procedure ConstructLocalCUPTree(i)
33 for each occurrence of node i in the
34 GlobalCUP-Tree
35 Initialize mappedTrans
36 for each hTWU item j ∈ LocalItemTable
37 in the path to the root
38 mappedTrans = mappedTrans ∪
39 GetIndex(j)
40 end for
41 Sort mappedTrans in ascending order of
42 item ids
43 InsertToCUPTree(mappedTrans)
44 end for
45 end for
46 Procedure RecMine(x)
47 for each child i of x
48 Set all TWU in LocalItemTable to 0
49 for each occurrence of node i in the
50 LocalCUPTree
51 for each item j in the path to the
52 root
53 Increment TWU of j in
54 LocalCUPTree
55 end for
56 end for
57 for each hTWU item k ∈ LocalItemTable
58 Calculate Real Utility and Attach k
59 as a child of i
60 end for
61 RecMine(i)
62 end for
63 end

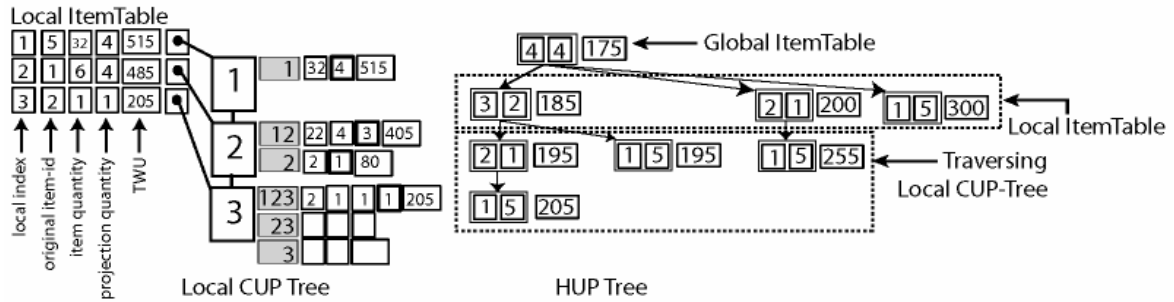
```

Figure 6. CTU-PRO Mining Algorithm

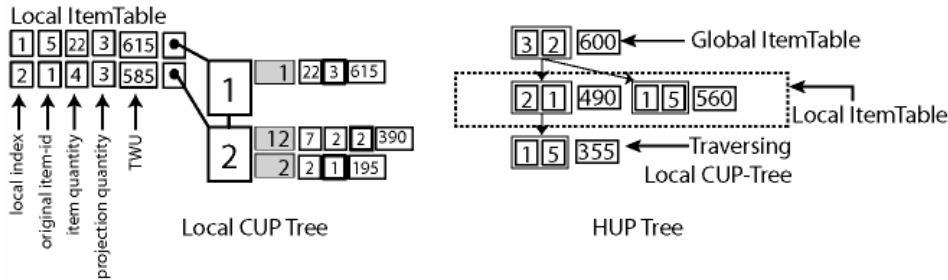
These local hTWU items are mapped to their indexes in the *LocalItemTable* on-the-fly (see Figure 5), sorted in ascending order of their index ids and inserted into the *LocalCUP-Tree* (lines 28-35). The first path of index 5 (identified by ❶) returns nothing. From the second path of index 5 (identified by ❷), a transaction 2 3 (mapped as 1 2 in *LocalItemTable*) with quantities 2 and 1 and TWU =195 is inserted into the *LocalCUP-Tree* along with quantity 1 for the projection of index 5.



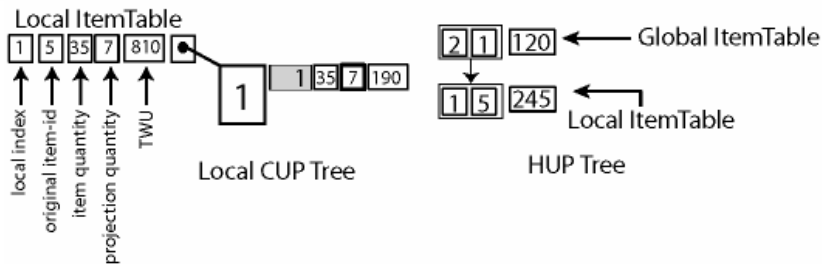
(a) Database projection index 5



(b) Database projection index 4



(c) Database projection index 3



(d) Database projection index 2

Figure 7. Mining on Database Projection

Another transaction from the third path (identified by ③), of index 5 (2 4, mapped as 1 3) with quantity 2 1 (and quantity 1 from projection 5) is also inserted with TWU=80. From the fourth path, index 2 (mapped as 1) with quantity 1 and 1, and TWU=35 are inserted. From the last path (identified by ⑤), item 4 (mapped as 3) and quantity 1, 1 and TWU=110 are inserted. Note that index 1 that represents original item-id 5 (quantity = 10) is not

inserted because it is not registered as hTWU in *LocalItemTable*. As the item indexes in the *GlobalItemTable* and *LocalItemTable* are different, the item id is always maintained for output purposes.

After visiting every node on the projection, the TWU and quantity of that node will be “folded” to its parent node. For example, after visiting the node identified by ③, the quantity and TWU of pattern 2 4 (i.e. 2 1 and

TWU=80) will be accumulated to the parent node, so that it can be used for the computation in the next loop.

**4.3. Mining from Projection Database.** Longer high utility itemsets, with length greater than two, are extracted by calling the procedure *RecMine* (line 12). For simplicity, we have described this procedure (lines 37-49) using recursion but, in the program, it is implemented as a non-recursive procedure. Starting from the least hTWU item in the *LocalItemTable* (line 38), the *node-link* is traversed (lines 40-44). For each node, the path to the root in the *LocalCUP-Tree* is traversed computing the other items that are together with the current item.

For example, in Figure 7a, traversing the *node-link* of item index 2 will return index 1, and since it is hTWU, a real utility value will be calculated by multiplying the appropriate quantity with the utility value in *GlobalItemTable*. The quantities at that node are 2 1 1 corresponding to the local pattern of 1 2 5 (original item ids = 1 2 3). So the utility value for the pattern would be  $(2 \times 10) + (1 \times 150) + (1 \times 25) = 195$ . An entry is created and attached as the child of index 2 in *HUP-Tree* (lines 45-47).

By traversing *HUP-Tree* (line 13) we can print the real utility of itemsets containing item 3 (index 5) as follows: 3 (100), 3 4 (120), 3 2 (175), 3 2 1 (195), 3 1 (125). Since the threshold is 129.9, only itemsets 3 2 and 3 2 1 are reckoned as high utility itemsets.

Figure 7b-d shows the *LocalCUP-Tree* and *HUP-Tree* for projection of indexes 4, 3 and 2. The mining process is continued with the next item in the *GlobalItemTable* for indexes 4, 3, 2 and finally, when the root of the tree in Figure 3 is reached, it outputs 5 (300).

The complete high utility itemsets (after mapping back to the original item names in Figure 1) are the following: {B(600), D(175), E(300), AB(490), AD(200), AE(245), BC(175), BD(185), BE(560), DE(300), ABC(195), ABD(195), ABE(355), ADE(255), BDE(195), ABDE(205)}.

## 5 Performance Study

In this Section, the performance of *CTU-PRO* is empirically compared with the implementation of *TwoPhase* downloaded from (CUCIS) and *CTU-Mine* (Erwin, Gopalan, & Achuthan, 2007). *CTU-PRO* is written in C++ and compiled using g++ version 4.1.0. All the testing was performed on a Pentium Core Duo, 3 GB RAM, running Linux Fedora Core 7 operating system.

We used the real datasets Retail and BMSPOS available from FIMI Repository (FIMI), in addition to several synthetic datasets. Retail is a market basket dataset from a Belgian supermarket, made available by Brijs et al. (Brijs, Goethals, Swinnen, Vanhoof, & Wets, 1999). The BMSPOS dataset (Kohavi, Brodley, Frasca, Mason, & Zheng, 2000) is point of sales data from a large electronics retailer, courtesy of Blue Martini Software.

We generated three synthetic datasets using our own program and IBM Quest data generator (IBM): (a) T5N5D100K with transaction length 5, size of maximal potentially frequent pattern 5 and the number of transactions 100K; (b) T10I5D100K, with transaction length 10, size of maximal potentially frequent pattern of

5 and number of transactions 100K, and (c) 50K transactions with a uniform length of 10.

Dataset	# of Trans	Average Length
Retail	88,162	10.3
BMSPOS	515,597	7.5
T5N5D100K	100,000	5
T10N5D100K	100,000	10
UNIFORM_10_50K	50,000	10

Table 1. Characteristics of Datasets

Table 1 shows the characteristics of the datasets. Since all these datasets are normally used for traditional frequent itemset mining, we had to add quantity and item utility values to the dataset. We generated a utility table based on lognormal distribution with the utility values ranging from 0.01 to 10. Figure 8, presents the frequency distribution of the utility values of 1000 items as an illustration. The quantities of items were generated randomly in the range of 1 to 10.

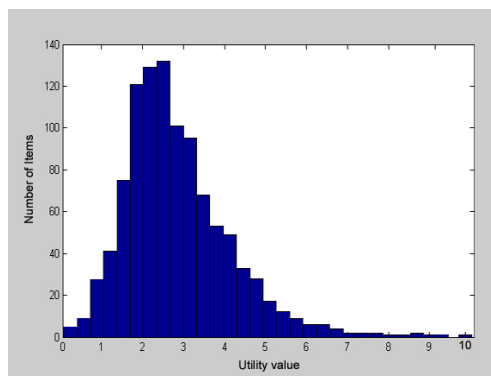


Figure 8. Distribution of utility value.

Results of our experiments are shown in Figure 9. It can be seen from the performance graphs that in general *CTU-PRO* outperforms *TwoPhase* and *CTU-Mine*. For high thresholds in the Retail dataset, *TwoPhase* runs slightly faster compared to *CTU-PRO*, but when the utility threshold becomes lower, *CTU-PRO* outperforms *TwoPhase*. For very low utility thresholds, the performance of *TwoPhase* got worse. This is due to the limitations of the generation-and-test approach of *TwoPhase* that has to traverse the database many times to enumerate and compute the large number of possible itemsets of high transaction weighted utility. As our algorithm is based on pattern growth using a compact transaction tree, repeated traversal of the database is avoided.

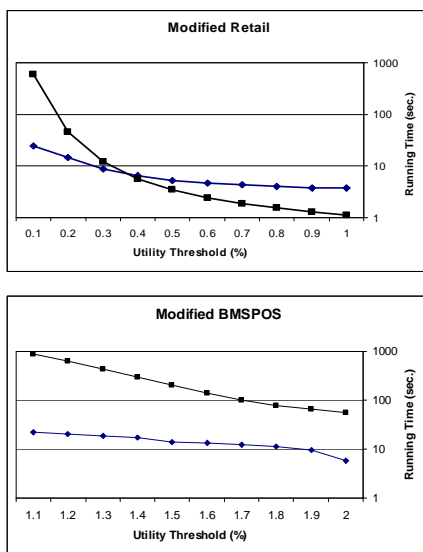
As the real datasets used were very sparse, the current implementation of *CTU-Mine* ran out of memory and so could not compute at the given support levels. Therefore, *CTU-PRO* is compared with only *TwoPhase* on the real datasets.

## 6 Conclusion

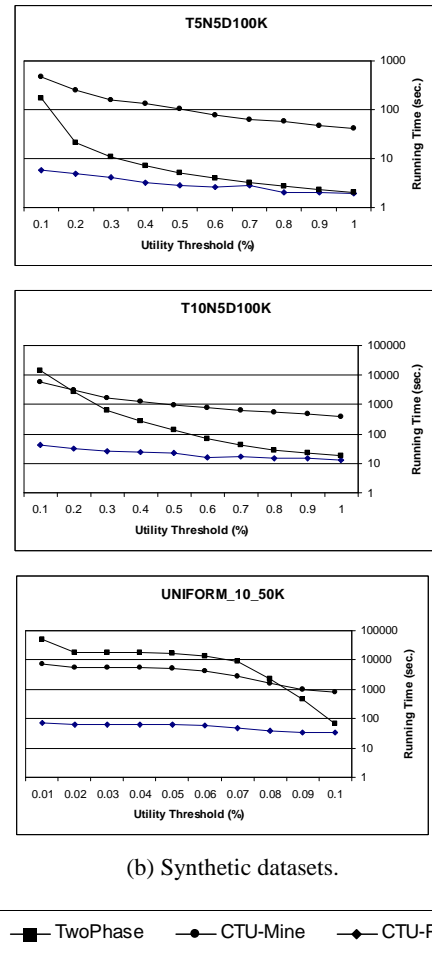
In this paper, we proposed the *CTU-PRO* algorithm to mine the complete set of high utility itemsets from both sparse and relatively dense datasets with short or long high utility patterns. The data structure and algorithm extend the pattern growth approach, taking into account the lack of anti-monotone property for pruning utility based patterns. We have compared the performance of *CTU-PRO* against the recent *TwoPhase* algorithm (Y. Liu, Liao, & Choudhary, 2005) and *CTU-Mine* (Erwin, Gopalan, & Achuthan, 2007). The results show that *CTU-PRO* works more efficiently than *TwoPhase* and *CTU-Mine* on dense data sets. *CTU-PRO* is also more efficient on sparse datasets at low support thresholds.

In our current algorithm, *GlobalCUP-Tree* is constructed entirely in the main memory. For very large databases, this approach is not adequate. We plan to investigate a parallel projection scheme using secondary storage as well as parallel processing on a shared-nothing cluster of computers as in (Sucahyo, Gopalan, & Rudra, 2003).

Another aspect to be studied is the overestimation of the TWU threshold. We used the same value of transaction weighted utility as the user specified itemset utility for growing and pruning the patterns. TWU is an overestimation of potential high utility itemsets, thus requiring more memory and computation compared to the pattern growth algorithms for frequent itemset mining that use exact values for pruning. Further research is needed to determine how the thresholds for transaction weighted utility may be varied from the user specified utility to reduce this overestimate. As the data for mining is very large in general, we also plan to study sampling based approximation (Lee, Cheung, & Kao, 1998; Toivonen, 1996; Mohammed Javeed Zaki, Parthasarathy, Li, & Ogihara, 1997) to make the computation less expensive.



(a) Real datasets.



(b) Synthetic datasets.

Figure 9. Execution time with varying minimum utility thresholds on real and synthetic datasets.

## 7 Acknowledgement

Alva Erwin is supported by Curtin International Research Tuition Scholarship (CIRTS). We thank Ying Liu for providing the *TwoPhase* program.

## 8 References

- Agrawal, R., Imielinski, T., & Swami, A. (1993, May). *Mining Association Rules between Sets of Items in Large Database*. Paper presented at the ACM SIGMOD International Conference on Management of Data, Washington DC.
- Agrawal, R., & Srikant, R. (1995). *Mining Sequential Patterns*. Paper presented at the 11th International Conference on Data Engineering.
- Brijs, T., Goethals, B., Swinnen, G., Vanhoof, K., & Wets, G. (1999). *The Use of Association Rules for Product Assortment Decisions: A Case Study*. Paper presented at the 5th International Conference on Knowledge Discovery and Data Mining, San Diego.
- CUCIS. Center for Ultra-scale Computing and Information Security, Northwestern University <http://cucis.ece.northwestern.edu/projects/DMS/MineBenchDownload.html>.

- Erwin, A., Gopalan, R. P., & Achuthan, N. R. (2007). *CTU-Mine: An Efficient High Utility Itemset Mining Algorithm Using the Pattern Growth Approach*. Paper presented at the IEEE 7th International Conferences on Computer and Information Technology, Aizu Wakamatsu, Japan.
- FIMI. Frequent Itemset Mining Implementations Repository <http://fimi.cs.helsinki.fi/>.
- Han, J., Pei, J., Mortazavi-Asl, B., Chen, Q., Dayal, U., & Hsu, M.-C. (2000). *FreeSpan: Frequent Pattern-Projected Sequential Pattern Mining*. Paper presented at the Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining Boston, Massachusetts, United States
- Han, J., Wang, J., & Yin, Y. (2000, May). *Mining frequent patterns without candidate generation*. Paper presented at the SIGMOD00.
- IBM. IBM Synthetic Data Generator <http://www.almaden.ibm.com/software/quest/resources/index.shtml>.
- Kohavi, R., Brodley, C., Frasca, B., Mason, L., & Zheng, Z. (2000). *KDD-Cup 2000 organizers' report: Peeling the onion*. Paper presented at the SIGKDD Explorations.
- Lee, S. D., Cheung, D. W., & Kao, B. (1998). Is Sampling Useful in Data Mining? A Case in the Maintenance of Discovered Association Rules. *Data Mining and Knowledge Discovery*, 233-262.
- Liu, B., Hsu, W., & Ma, Y. (1998). *Integrating Classification and Association Rule Mining*. Paper presented at the Knowledge Discovery and Data Mining, New York.
- Liu, Y., Liao, W.-K., & Choudhary, A. (2005). *A Fast High Utility Itemsets Mining Algorithm*. Paper presented at the 1st international workshop on Utility-Based Data Mining (UBDM), Chicago Illinois.
- Sucahyo, Y. G., & Gopalan, R. P. (2004). *Building a More Accurate Classifier Based on Strong Frequent Patterns*. Paper presented at the Australian Joint Conference on Artificial Intelligence.
- Sucahyo, Y. G., & Gopalan, R. P. (2004, Nov 1-4). *CT-PRO: A Bottom-Up Non Recursive Frequent Itemset Mining Algorithm Using Compressed FP-Tree Data Structure*. Paper presented at the IEEE ICDM Workshop on Frequent Itemset Mining Implementation (FIMI), Brighton UK.
- Sucahyo, Y. G., Gopalan, R. P., & Rudra, A. (2003). Efficiently Mining Frequent Patterns from Dense Datasets Using a Cluster of Computers. In *AI 2003: Advances in Artificial Intelligence* (Vol. Volume 2903/2003, pp. 233-244): Springer Berlin / Heidelberg.
- Toivonen, H. (1996). *Sampling Large Databases for Association Rules*. Paper presented at the 22nd VLDB Conference, Bombay.
- Yao, H., & Hamilton, H. J. (2006). Mining itemset utilities from transaction databases. *Data & Knowledge Engineering*, 59(3), 603 - 626.
- Yao, H., Hamilton, H. J., & Buzz, C. J. (2004). *A Foundational Approach to Mining Itemset Utilities from Databases*. Paper presented at the 4th SIAM International Conference on Data Mining, Florida USA.
- Yao, H., Hamilton, H. J., & Geng, L. (2006). *A Unified Framework for Utility Based Measures for Mining Itemsets*. Paper presented at the 2nd Workshop on Utility-Based Data Mining (UBDM).
- Zaki, M. J. (2000). Scalable Algorithms for Association Mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3).
- Zaki, M. J., Parthasarathy, S., Li, W., & Ogihara, M. (1997). *Evaluation of Sampling for Data Mining of Association Rules*. Rochester, New York: University of Rochester.