Efficient Mining of High Utility Itemsets from Large Datasets

Alva Erwin[1], Raj P. Gopalan[1], and N.R. Achuthan[2]

[1] Department of Computing, [2] 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. High utility itemsets mining extends frequent pattern mining to discover itemsets in a transaction database with utility values above a given threshold. However, mining high utility itemsets presents a greater challenge than frequent itemset mining, since high utility itemsets lack the *anti-monotone* property of frequent itemsets. Transaction Weighted Utility (TWU) proposed recently by researchers has *anti-monotone* property, but it is an overestimate of itemset utility and therefore leads to a larger search space. We propose an algorithm that uses TWU with pattern growth based on a compact utility pattern tree data structure. Our algorithm implements a parallel projection scheme to use disk storage when the main memory is inadequate for dealing with large datasets. Experimental evaluation shows that our algorithm is more efficient compared to previous algorithms and can mine larger datasets of both dense and sparse data containing long patterns.

Keywords: High Utility Mining, Pattern Growth.

1 Introduction

The goal of frequent itemset mining [1] is to find items that co-occur in a transaction database above a user given frequency threshold, without considering the quantity or weight such as profit of the items. However, quantity and weight are significant for addressing real world decision problems that require maximizing the utility in an organization. The high utility itemset mining problem is to find all itemsets that have utility larger than a user specified value of minimum utility. Yao et al [2, 3] proposed a framework for high utility itemset mining. Recent research has focused on improving the efficiency of high utility mining. Liu et al. [4] proposed a *TwoPhase* algorithm based on *Apriori* [1] to mine high utility itemsets, using a transaction weighted utility (TWU) measure to prune the search space. Their algorithm is suitable for sparse data sets with short patterns. We recently developed an algorithm named *CTU-Mine* [5] based on the pattern growth approach [6] that was efficient on dense data with relatively longer patterns. In [7], we proposed an algorithm named *CTU-PRO* for efficient mining of both dense and sparse data sets that fit into main memory.

In this paper, we propose an algorithm named *CTU-PROL* for mining high utility itemsets from large datasets using the pattern growth approach [6]. The algorithm first identifies the large TWU items in the transaction database and if the dataset is relatively

T. Washio et al. (Eds.): PAKDD 2008, LNAI 5012, pp. 554–561, 2008.

small, it creates a *Compressed Utility Pattern Tree* (*CUP-Tree*) for mining high utility itemsets. For data sets too large to be held in main memory, the algorithm creates subdivisions using parallel projections that can be subsequently mined independently. For each subdivision, a *CUP-Tree* is used to mine the complete set of high utility itemsets. The *anti-monotone* property of TWU is used for pruning the search space of subdivisions in *CTU-PROL*, but unlike *TwoPhase* of Liu et al. [4], our algorithm avoids a rescan of the database to determine the actual utility of high TWU itemsets. The performance of *CTU-PROL* is compared against the implementation of the *TwoPhase* algorithm [4] available from [8] and also with *CTU-Mine* [5]. The results show that *CTU-PROL* outperforms previous algorithms on both sparse and dense datasets at most support levels.

2 Terms and Definitions

In this Section, we define the basic terms of high utility itemset mining based on [1, 2, 9]. Let $I=\{i_1,i_2,\ldots,i_m\}$ be a set of items and $D=\{T_1,T_2,\ldots,T_n\}$ 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:

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

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

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

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. [4] 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:

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

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

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

As shown in [4], any superset of a low TWU itemset is also a low TWU itemset, and so we can prune all supersets of low TWU itemsets. However, since TWU is an over-estimation of the real utility value, high TWU itemsets will need to be pruned further. Consider a transaction database of a retailer as shown in Fig. 1(a), with external utility of each item as in Fig. 1(b). The values in each row of Fig. 1(a) shows the quantities of items bought in a transaction, and the last column contains the transaction utility with the total transaction utility of the database shown in the last row. From this sample data, we can compute, $u(3\ 4, t_1)$ = \$60, $u(3\ 4, t_3)$ = \$60, $u(3\ 4, t_5)$ = \$60, $u(3\ 4)$ = \$180 and $twu(3\ 4)$=\$262.

TID	1	2	3	4	5	6	TU.
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
total	12	4	4	5	60	2	1299

(a) Transaction Database

		Profit ($)
1	Printer Ink	10
2	Colour Laser Printer	150
3	Bubble Jet Printer	25
4	Digital Camera	35
5	Glossy Photo Paper	5
6	Floppy Disk	2

(b) Utility Table

Fig. 1. An example transaction database and utility table

3 Mining High Utility Itemsets in Large Datasets

In this section, we describe the *CTU-PROL* algorithm for mining large datasets, consisting of the following steps: (1) Create a *GlobalItemTable* by scanning the database to identify items of high TWU, (2) Subdivide the database by parallel projection of transactions with high TWU items, and (3) Mine each subdivision for high utility itemsets after constructing a Compressed Utility Pattern-Tree (*CUP-Tree*) for the subdivision.

3.1 Creating *Global ItemTable*

A *GlobalItemTable* is constructed as explained in [5] and maps the item-ids to integers in the descending order of their TWU values. Fig. 2 gives the *GlobalItemTable* for the database of Fig. 1, with a minimum utility of 10% of the total transaction utility (=129.9). Note that item 6 with TWU of 99 is pruned. The mapped new index of 1 to 5 correspond to the original items 5, 1, 2, 4, and 3 respectively. The terms mapped item id and item index are used synonymously in the rest of the paper.

GlobalItemTable

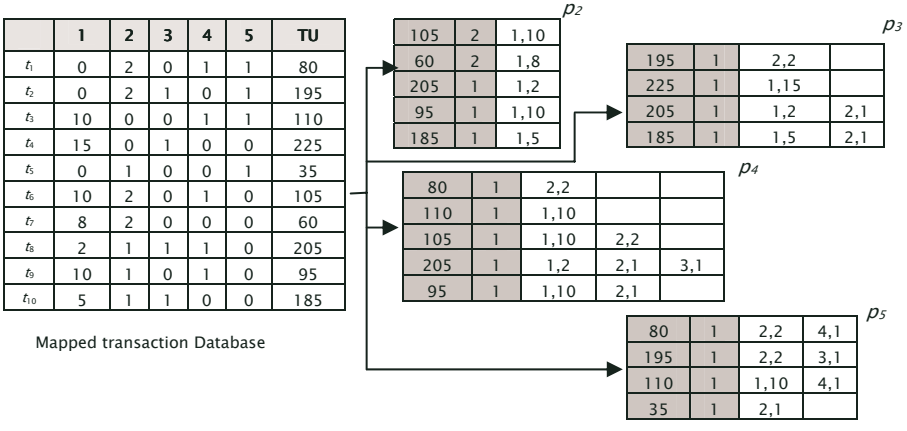
Item index	1	2	3	4	5
Orginal item id	5	1	2	4	3
Profit	5	10	150	35	25
Quantity	60	12	4	5	4
TWU	987	964	810	595	422

Fig. 2. *GlobalItemTable* of database of Fig. 1

3.2 Database Subdivision by Parallel Projection

We adapt the concept of parallel projection reported in [10] for datasets that are too large for the corresponding *CUP-Tree* to fit into main memory. Using the *GlobalItemTable* of Fig. 2, the original database is transformed into the mapped transaction database. Concurrently the parallel projection scheme is constructed. Fig. 3

illustrates the process for the database of Fig. 1 using the corresponding *GlobalItemTable*. For item index $i \geq 2$, every transaction t with positive quantity of items up to item i is written into the subdivision p_i with the corresponding quantities and TWU. So there are five entries in p_2 from transactions t_6, t_7, t_8, t_9 and t_{10}. Similarly, in p_3, we have entries for item index 1 and/or 2 that occur before index 3 in the transactions t_2, t_4, t_8 and t_{10}. The third subdivision (p_4) is from transactions t_1, t_3, t_6 t_8, and t_9, and the last subdivision from transactions t_1, t_2, t_3, and t_5.



	1	2	3	4	5	TU
t_1	0	2	0	1	1	80
t_2	0	2	1	0	1	195
t_3	10	0	0	1	1	110
t_4	15	0	1	0	0	225
t_5	0	1	0	0	1	35
t_6	10	2	0	1	0	105
t_7	8	2	0	0	0	60
t_8	2	1	1	1	0	205
t_9	10	1	0	1	0	95
t_{10}	5	1	1	0	0	185

105	2	1,10	
60	2	1,8	
205	1	1,2	
95	1	1,10	
185	1	1,5	

195	1	2,2	
225	1	1,15	
205	1	1,2	2,1
185	1	1,5	2,1

80	1	2,2		
110	1	1,10		
105	1	1,10	2,2	
205	1	1,2	2,1	3,1
95	1	1,10	2,1	

80	1	2,2	4,1
195	1	2,2	3,1
110	1	1,10	4,1
35	1	2,1	

Fig. 3. Parallel projection of transaction database

Once the subdivisions are created, each subdivision p_i is mined separately for high utility patterns.

3.3 Mining Subdivisions Using Compressed Utility Pattern Tree

The Compressed Utility Pattern Tree (*CUP-Tree*) originally proposed by us in [7] is a variant of *CFP-Tree* [11] and *CTU-Tree* [5] data structures. In this paper for a given subdivision p_i of the database, we construct the corresponding *Compressed Utility Pattern Tree* (*CUP-Tree*) exactly in the same way as in [5]. Fig. 4 illustrates the *CUP-Tree* and the *GlobalItemTable* for subdivision p_5 of the database given in Fig. 1. More explicitly, the first row of subdivision p_5 has mapped item-ids (pattern) 2, 4 and 5 (for the original items 1, 3 and 4) with respective quantities 2, 1, and 1. This pattern is inserted into the tree with the TWU (80) and a pointer to the array of quantities 2, 1 and 1 (node labeled ❸ in Fig. 4). The nodes which represent the current subdivision p_5 (index 5) are linked by node links to facilitate the traversal in the mining process. All the transactions will be inserted similarly, giving the *CUP-Tree* of Fig. 4.

Now mining for the subdivision p_5 is initiated using the *CUP-Tree* of Fig. 4 as input. Traversing the nodelink of index 5 (Fig. 4.) the associated items are recorded in the projection tree named *ProCUP-Tree*. The information for extracting high utility items is recorded in a *High Utility Pattern Tree (HUP-Tree)* with mapped item id 5 as its root (labeled A in Fig. 6 which shows the *ProCUP-Tree* and *HUP-Tree* of subdivision p_5). The mining of a subdivision consists of three steps: (1) Construction of

ProItemTable, (2) Construction of *ProCUP-Tree*, and (3) Mining by traversing *ProCUP-Tree*. These steps are explained below using subdivision p_5 as an example.

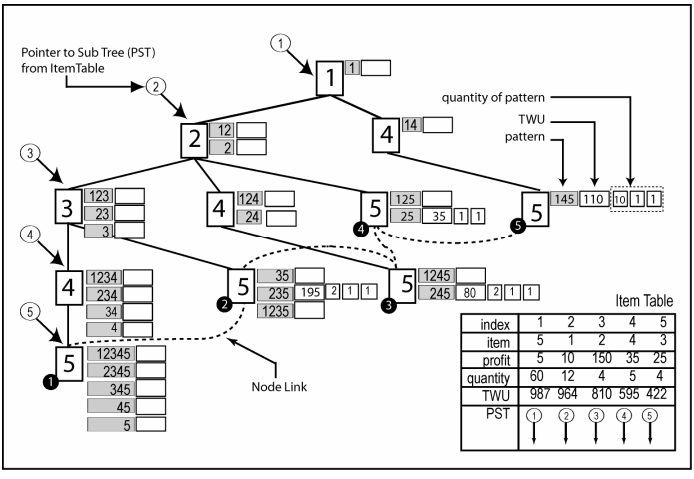


Fig. 4. *CUP-Tree* and *ItemTable* for projection p_5

Step 1. Construction of Pro*ItemTable*. Traversing the node link of index 5 (Fig. 4), the associated items are arranged as *ProItemTable* in the descending order of the TWU values. This table is constructed in the same manner as the *GlobalItemTable* explained in [5] restricting attention to linked nodes of index 5 in p_5.

For example, traversing of the linked nodes provides the following indexes and TWU values column 1 (110) from ❺, 2 (310) from (❷,❸,❹), 3 (195) from ❷, and 4 (190) from (❸, ❺). Since our *min_utility* is 129.9, index 1 is pruned and indexes 2,3,4 (item ids: 1,2,4) are locally hTWU in ProItemTable (see Fig. 6). The mapping of the *GlobalItemTable* item index to the *proItemTable* item index and the corresponding original item-ids are provided in Fig. 5. Concurrently the level 1 children of the *HUP-Tree* root are recorded (indicated by label B in Fig. 6). Furthermore, note that the *proItemTable* includes a column giving the cumulative quantity of the projection item 5.

***GlobalItemTable* index**	1	2	3	4	5
Original item index	5	1	2	4	3
***ProItemTable* index**	–	1	2	3	–

Fig. 5. Mapping projection of index 5 using ProItemTable

Step 2. Construction of *ProCUP-Tree*. Retraversing the *node-link* in the *CUP-Tree* and using the mapping in *ProItemTable* (see Fig. 5), we construct the *ProCUP-Tree.* Note that *ProCUP-Tree* is expressed using the *proItemTable* index.

Step 3. Mining by *ProCup-Tree* Traversal. For each item in *proItemTable*, the path to the root is traversed computing the other items that are together with the current item. In Fig. 6, traversing the *node-link* of item index 2 will return index 1, and since it

has high TWU, the real utility value will be calculated by multiplying the appropriate quantity with the utility value in *GlobalItemTable*. The quantity at that node is 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 (indicated by label C in Fig. 6) in the *HUP-Tree.*

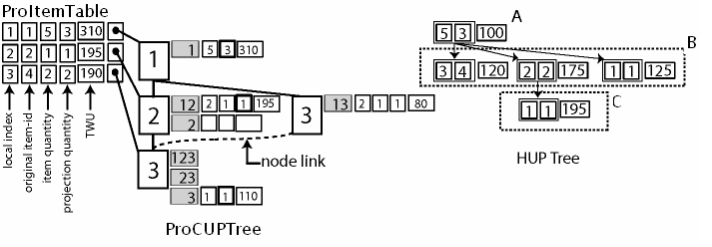


Fig. 6. ProCUPTree and HUP Tree

By traversing the *HUP-Tree* 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. The mining process is continued with the remaining subdivisions p_4, p_3, p_2. The complete high utility itemsets (after mapping back to the original item-ids) are obtained as the following: {2(600), 4(175), 5(300), 12(490), 14(200), 15(245), 23(175), 24(185), 25(560), 45(300), 123(195), 124(195), 125(355), 145(255), 245(195), 1245(205)}.

4 Performance Study

In this Section, the performance of *CTU-PROL* is empirically compared with the implementation of *TwoPhase* downloaded from [8] and *CTU-Mine* [5]. *CTU-PROL* is written in C++ and compiled using g++ version 4.1.0. The experiments were performed on a Pentium Core Duo, 3 GB RAM, with Linux operating system. We used the real datasets Retail and BMSPOS available from the FIMI Repository [12]. We also generated the synthetic datasets T10N5D100K and T5N5DXM using our program and IBM Quest data generator [13] to test the scalability of our algorithm. 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 datasets. We generated utility values from a suitable log-normal distribution, and the quantities randomly from numbers one to ten.

Results of our experiments are shown in Fig. 7. For high thresholds in the Retail dataset, *TwoPhase* runs slightly faster compared to *CTU-PROL*, but when the utility threshold becomes lower, *CTU-PROL* 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 tree, repeated traversal of the database is avoided.

Table 1. Characteristics of Datasets

Dataset	# of Trans	# of different. item	Size of file (MB)
Modifed Retail	88,162	1,658	8
Modifed BMSPOS	515,597	16,470	30
T10N5D100K	100,000	100	5
T5N5DXM	1,000,000 – 5,000,000	100	35 – 200

Note that for relatively large datasets, *CTU-Mine* ran out of memory and hence *CTU-PROL* is compared with only *TwoPhase.* On the synthetic dataset T10N5D100K, we tested *CTU-Mine*, *CTU-PROL* and *TwoPhase*. When the utility threshold is low, *TwoPhase* is unable to complete within a 10,000 seconds time limit. For scalability, we tested *TwoPhase* and *CTU-PROL* using T10N5DXM datasets with *min_utility* 0.05% of total utility. In general, the results show that *CTU-PROL* outperforms *CTU-Mine* and *TwoPhase* for various utility thresholds and transaction volumes.

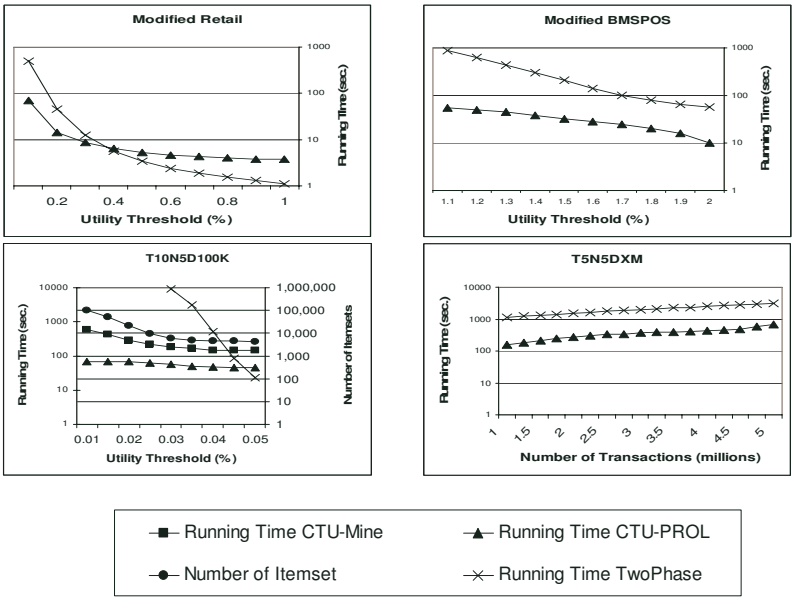


Fig. 7. Execution time with varying minimum utility thresholds and number of transactions on real and synthetic datasets

5 Conclusion

In this paper, we have presented the *CTU- PROL* algorithm to mine the complete set of high utility itemsets from both sparse and relatively dense datasets with short or longer high utility patterns. Our 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-PROL* against the recent *TwoPhase* algorithm [4] and *CTU-Mine* [5]. The results show that *CTU-PROL*

works more efficiently than *TwoPhase* and *CTU-Mine*. Our algorithm adapts to large data by constructing parallel subdivisions on disk that can be mined independently. The experiments show that *CTU-PROL* is scalable for larger datasets.

Since TWU is an overestimation real utility, resources used are possibly high for these pattern growth algorithms. Further research is needed to determine how the thresholds for TWU may be varied from the user specified utility to reduce this overestimate. As the data for mining is very large in general, we plan to study sampling based approximations to reduce the computation.

Acknowledgement

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

References

1. Agrawal, R., Imielinski, T., Swami, A.: Mining Association Rules between Sets of Items in Large Database. In: ACM SIGMOD International Conference on Management of Data (1993)
2. Yao, H., Hamilton, H.J., Buzz, C.J.: A Foundational Approach to Mining Itemset Utilities from Databases. In: 4th SIAM International Conference on Data Mining. Florida USA (2004)
3. Yao, H., Hamilton, H.J.: Mining itemset utilities from transaction databases. Data & Knowledge Engineering 59(3), 603–626 (2006)
4. Liu, Y., Liao, W.K., Choudhary, A.: A Fast High Utility Itemsets Mining Algorithm. In: 1st Workshop on Utility-Based Data Mining. Chicago Illinois (2005)
5. Erwin, A., Gopalan, R.P.: N.R. Achuthan.: CTU-Mine: An Efficient High Utility Itemset Mining Algorithm Using the Pattern Growth Approach. In: IEEE CIT 2007. Aizu Wakamatsu, Japan (2007)
6. Han, J., Wang, J., Yin, Y.: Mining frequent patterns without candidate generation. In: ACM SIGMOD International Conference on Management of Data (2000)
7. Erwin, A., Gopalan, R.P., Achuthan, N.R.: A Bottom-Up Projection Based Algorithm for Mining High Utility Itemsets. In: International Workshop on Integrating AI and Data Mining. Gold Coast, Australia (2007)
8. CUCIS. Center for Ultra-scale Computing and Information Security, Northwestern University, `http://cucis.ece.northwestern.edu/projects/DMS/MineBenchDownload.html`
9. Yao, H., Hamilton, H.J., Geng, L.: A Unified Framework for Utility Based Measures for Mining Itemsets. In: ACM SIGKDD 2nd Workshop on Utility-Based Data Mining (2006)
10. Pei, J.: Pattern Growth Methods for Frequent Pattern Mining. Simon Fraser University (2002)
11. Sucahyo, Y.G., Gopalan, R.P.: CT-PRO: A Bottom-Up Non Recursive Frequent Itemset Mining Algorithm Using Compressed FP-Tree Data Structure. In: IEEE ICDM Workshop on Frequent Itemset Mining Implementation (FIMI). Brighton UK (2004)
12. FIMI, Frequent Itemset Mining Implementations Repository,
13. `http://fimi.cs.helsinki.fi/`
14. IBM Synthetic Data Generator, `http://www.almaden.ibm.com/software/quest/resources/index.shtml`