AN EUP_TREE-BASED ALGORITHM FOR MINING MAXIMIZED ERASABLE UTILITY PATTERNS IN THE INCREMENTAL TRANSACTIONAL DATABASE

Ye-In Chang [1]****, Po-Chun Chuang*** [1]***, Xuan-Hong Lin*** [1]***, Xiang-Long Ding*** [1]

[1] *Department of Computer Science and Engineering, National Sun Yat-Sen University, Kaohsiung, Taiwan*
** E-mail: changyi@mail.cse.nsysu.edu.tw*

Keywords: Data Mining, Erasable Patterns Mining, Incremental Database, Maximized Erasable Utility, Tree Structure-Based Approach

Abstract

The erasable pattern mining is a technique for finding patterns with profits not higher than the threshold. However, in the real world, we need to consider both the value and the quantity of products in each transaction. Moreover, we also extract the maximized erasable utility by considering the utility of a pattern. This topic can find patterns with the low external utility. One of those well-known algorithms to achieve this goal is the *MEL* algorithm. However, it needs to scan the database twice. Therefore, to improve the performance, we propose an efficient algorithm called the *MEUPM* algorithm, which just needs to scan the database once. The proposed algorithm achieves the goal by using two tree structures and two pruning strategies. From our performance study, we have shown that the performance of our algorithm is better than that of the *MEL* algorithm

1 Introduction

Erasable pattern mining aims to identify low-value patterns whose removal can improve profitability, differing from traditional approaches that emphasize frequent or high-value patterns. A pattern X is evaluated by its *Gain(X)*, defined as the sum of profits of products containing subsets of *X*. A pattern is erasable if its gain does not exceed the maximum gain threshold (*MGT*), computed as a user-defined ratio of the total product profit.

Incremental erasable pattern mining was introduced to avoid re-scanning the entire database whenever new transactions arrive. Existing incremental methods rely on list-based [1][2][3][4] or tree-based structures [5]. Lee *et al*. [6] further extended the task to maximized erasable utility patterns, integrating internal and external utilities. They replace *Gain* with *MGain*, and a pattern is considered a maximized erasable utility pattern when *MGain* ≤ *MGT*.

Research in erasable pattern mining can be grouped into five categories: weighted erasable patterns [1][3][7], erasable closed patterns [8][9][10], erasable patterns over uncertain data [11][12], incremental erasable patterns [2][13], and maximized erasable (utility) patterns [6][14]. Among them, the *MEL* algorithm [6] is a representative method for mining maximized erasable utility patterns. However, *MEL* is designed for static databases; each data insertion forces a complete re-scan and a list-sort operation, leading to high computation cost.

To address these limitations, we propose the *MEUPT* algorithm for mining maximized erasable utility patterns in both static and incremental environments. Our method adopts a tree structure to store database and update information, requires only a single scan of the original database, and incorporates efficient update handling for new transactions. In addition, multiple pruning strategies are applied to eliminate unpromising candidates early. Experimental results demonstrate that *MEUPT* substantially outperforms the *MEL* algorithm.

2. Methodology

Our algorithm employs two trees. The *EUP_tree* stores information from the original database, while the *MEUPM_tree* maintains erasable items. The procedure consists of two main stages. First, transactions are inserted into the *EUP_tree* and the header table is updated. Second, after inserting all transactions, items are sorted in descending support order to obtain erasable items, which are recorded in the *EI-list*. The *EUP_tree* is then reconstructed according to this order, and erasable nodes are inserted into the *MEUPM_tree*. Traversing this tree produces the order table and pattern table, which guide the pattern-expansion phase. Two pruning strategies are applied to accelerate the mining of maximized erasable utilitypatterns. The tree structure enables efficient computation of *GainV*. For incremental data, only the affected parts of the *EUP_tree* are updated.

2.1 Figures

The algorithm relies on two core structures: The Pattern Table, which stores the pre-order of items for pattern expansion. The Order Table, which records the pre- and post-order of each node in the *MEUPM_tree* for determining node relationships.

2.1.1 Example of a Transactional Database: A transactional database stores the internal utility (*iu*) of each item per transaction, while external utilities (*eu*) are stored in an external utility table. Figure 1 shows an example where each transaction lists its items and internal utilities. External utilities for items *A*–*I* are 4, 2, 3, 6, 1, 2, 3, 4, and 2, respectively.

2.1.2 Parameter Definitions: For each transaction *Tn*, its profit is stored in the Profit Table (*PT*) and computed as fallows:

$$Prof(T_n) = \sum_{i \subseteq T_n} eu(i) \cdot iu(i, T_n).$$

For example, in Figure 1, *prof*(T_1) = 26. The *PT* values for T_1–T_7 are 26, 31, 54, 31, 72, 10, and 101. The external utility of a pattern is the sum of its items' external utilities; *e.g.*, *eu(AB)* = 6. The maximum gain threshold (*MGT*) is defined as δ times the total profit of the database. With δ = 0.55, the *MGT* for Figure 1 is 178.75

TID	$Items$
T_1	A$(2), F(3), G(4)$
T_2	$B(4), C(4), E(7), F(2)$
T_3	A$(5), D(3), H(4)$
T_4	C$(3), E(4), G(6)$
T_5	C$(10), D(3), H(6)$
T_6	$F(3), I(2)$
T_7	$A(20), G(7)$

Fig. 1 The transactional database *TDB*

The Gain Value *X* (*GainV(X)*) represents the sum of *prof* influenced by this pattern *X* in transactional database *TDB*. *GainV* is the value for traditional erasable pattern mining to determine whether the pattern is an erasable pattern or not. For example, *GainV(A)* is 181. The Maximized Gain Value *MGainV* of a pattern *X* is an important value, which compares itself with *MGT* to determine whether pattern *X* is the maximized erasable pattern or not. For example, *MGainV(A)* is 724 Because the *MGainV* of item *A* is larger than *MGT* (= 174.5), item *A* is non-erasable.

2.1.3 An Example to Illustrate Construction of the EUP_tree: We propose the data structure which is similar to the *FP-tree* structure. The *Header Table* is used to record the occurrence of items, along with their support and *GainV* in the transaction. The Tree Node stores the *PreOrder* and *GainV* of the item. The initial value of pre-order in the node is null. We will change the value of pre-order after the reconstruction of the *EUP_tree*. The *Order Table* stores the pre-order, post-order and *GainV* of Tree Nodes in the *EUP_tree*, as shown in Figure 2. The *Pattern Table* stores the pattern and the set of pre-order where the pattern exists.

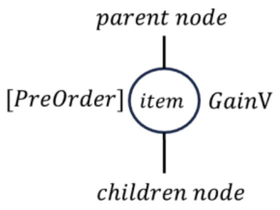


Fig. 2 The Tree Node

Take Figure 1 as an example to illustrate how we construct the *EUP_tree* as shown in Figure 3. Before database *TDB* is scanned, we first create the root node and initialize the *Header Table*. The item of the root node is an empty set, and both *GainV* and *PreOrder* are null. When the first transaction T_1 is scanned, we calculate the profit of transaction T_1 and update the Header Table for the items in transaction T_1 = *{A, F, G}*. Then, we create the three nodes for transaction T_1 and insert those nodes into the *EUP_tree*. Here path *AFG* is the first branch of the root. In fact, the right part of nodes *A*, *F*, *G* has value 26 (*i.e.*, the *Gain value*) (Note that although the right part of node *A* has value 181, which is the summing of *Gain* values of those parts following from the node *A*.) Next, when transaction T_2 = *B, C, E, F* is scanned, we again compute its profit and update the *Header Table*. Since item *F* has already appeared in T_1, its *GainV* and support are updated to *prof*(T_1) + *prof*(T_2) (= 26 + 31 = 57) in the related *Header Table*. After that, four new nodes are created for T_2 and inserted into the *EUP_tree* as the path. Continuing with transaction T_3 = *A, D, H,* we update the *Header Table* accordingly. Because item *A* already exists in the *EUP_tree*, its *GainV* is updated, and since the subsequent item *D* does not share the same branch as *F*, the nodes *D* and *H* are newly inserted. This is, we have a new branch from node *A* which a new path *DH* (with the *GainV* value = 54). Finally, the *GainV* of item *A* becomes *prof*(T_1)+*prof*(T_3) (= 26 + 54 = 80). For illustration of the overall structure after further insertions, Figure 3 contains the Header Table and the *EUP_tree* after transaction T_7. In fact, the contains of the *Header Table* are as follows: [item, support, *Gain*] = [*A*, 3, 181], [*F*, 3, 67], [*G*, 3, 158], [*B*, 1, 31], [*C*, 3, 134], [*E*, 2, 62], [*D*, 2, 126], [*H*, 2, 126], [*I*, 1, 10].

2.2 The Mining Process

Now, we will describe our mining process. There are two parts in our mining process. The first part is to mine the maximized erasable utility patterns in the original database. We will reconstruct the *EUP_tree* and construct the *MEUPM_tree*. According to the *MEUPM_tree*, we generate the *Order Table* and the *Pattern Table*. We will mine the maximized erasable utility patterns based on the *Order Table* and the *Pattern Table*. The second part is the strategy for mining the maximized erasable utility patterns after new data is inserted. When the new transactions are inserted, we will update the *EUP_tree*.

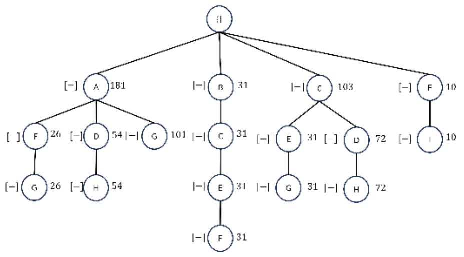


Fig. 3 The EUP_tree after transcation T_7 is inserted

2.2.1 The Reconstruction of the EUP_tree and Construction of the MEUPM_tree: After transactions in database *TDB* are inserted into *EUP_tree*, we will restructure the *EUP_tree* as shown in Figure 4 and construct the *MEUPM_tree* as shown in Figure 5. First, we sort the *Header Table* in the descending order based on supports of items. The *Header Table* after the sorting step shows that items *A*, *F*, *G*, and *C* all have support value 3 with corresponding *Gain* values of 181, 67, 158, and 134, respectively. Items *E*, *D*, and *H* have support value 2 with *Gain* values of 62, 126, and 126, respectively. Items *B* and *I* have support value 1 with *Gain* values of 31 and 10, respectively. The order is [*A*, *F*, *G*, *C*, *E*, *D*, *H*, *B*, *I*]. We can get the maximized erasable utility item and store them in the *Pattern Table*, when we sort the *Header Table*. The maximized erasable utility items are items *F*, *E*, *B*, *I*. Second, we reconstruct the *EUP_tree* according to the order in the *Header Table*. We assign the nodes with the high support as parent nodes for each branch in the *EUP_tree*. We use the Depth-First-Search strategy to traverse to the leaf node and find the branch.

For example, the first branch is nodes *A*, *F*, *G* and the *GainV* of the leaf node *G* is 26. We create the same roots for the *EUP_tree* and the *MEUPM_tree*. We sort these three items (*A*, *F*, *G*) by the order in the *Header Table* and the *GainV* of nodes is 26. Then, we insert them (*A*, *F*, *G*) into the *EUP_tree*. At the same time, we construct the *MEUPM_tree*. In this branch, node *F* is the maximized erasable utility item. We set the *GainV* of node *F* to 26 and insert it into the *MEUPM_tree*.

Next, the *GainV* of all nodes in this path minus the *GainV* of the leaf node *F*. The *GainV* of node *A* is 155 (= 18126). We delete nodes *F* and *G* because the *GainV* of nodes *F* and *G* are 0. The second branch is nodes *A*, *D*, *H* and the *GainV* of leaf node *H* is 54. We sort these three items (*A*, *D*, *H*) by the order in the *Header Table* and the *GainV* of nodes is 54. Because the prefix node *A* is already in the new *EUP_tree*, we will update the *GainV* of node *A* and insert the remaining nodes *D*, *H*. Because there is no maximized erasable utility item in this branch, we do not need to update the *MEUPM_tree*. Next, the *GainV* of all nodes in this path minus the *GainV* of the leaf node *G*. The *GainV* of node *A* is 101 (= 155- 54). We delete the nodes *D* and H, because the *GainV* of nodes *D* and *H* are 0. There are nodes *B*, *C*, *E*, *F* in the fourth branch. We arrange them as *F*, *C*, *E*, *B* and insert them into the *EUP_tree*. Items *B*, *E*, *F* are the maximized erasable utility items so we arrange them as *F*, *E*, *B* by their support order and insert them into the *MEUPM_tree*. Figure 4 shows *EUP_tree* after the construction. Figure 5 shows *MEUPM_tree* after the construction. (Note that the *EUP_tree* show in Figure 4, for example, node *F* with the total Gain value = 6(=26 + 31 + 10) which is the summation og the Gain values of transaction *T1*, *T2* and *T6*. Because those three transactions contain pattern F. Moreover, the value 67 is smaller than the threshold *MGT* = 178.75; therefore it is the result.)

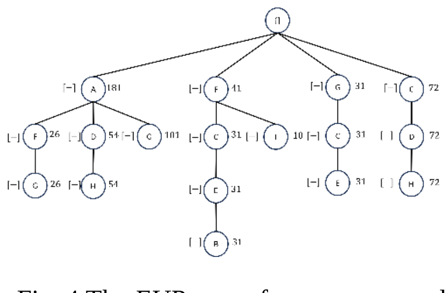


Fig. 4 The *EUP_tree* after reconstructed

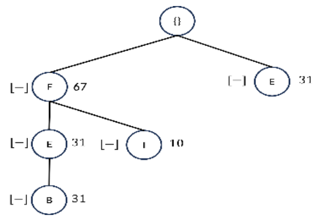


Fig. 5 The *MEUPM_tree* after constructed

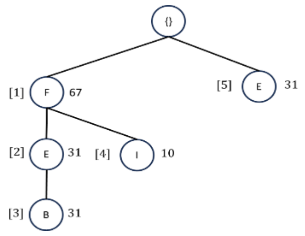


Fig. 6 The *MEUPM_tree* after setting the *PreOrder* of nodes

Preorder	1	2	3	4	5
Postorder	4	2	1	3	5
GainV	67	31	31	10	31

Fig. 7 The *Order Table* for the *MEUPM_tree*

After we construct the *MEUPM_tree*, we do the preorder traversal to set the *PreOrder* of nodes in the *MEUPM_tree* and construct the *Pattern Table*. Figure 6 shows the *MEUPM_tree* after the preorder traversal. We record the pre-order in the set for items and store them in the *Pattern Table*. For example, in Figure 6, the item *E* exists in nodes which preorder are 2 and 5. Figure 8 shows the *Pattern Table* of nodes in the *EUP tree*. Next, we do the postorder traversal to construct the *Order table* by the *PreOrder* of nodes in *MEUPM_tree*. We record the post-order and *Gain* value of nodes and store them in the *Order table*. For example, in Figure 6, the post-order of pre-order 1 is 4 and its *GainV* is 67. Figure 7 shows the *Order Table* of nodes in the *MEUPM tree*. (Note that in Figure 7, it stores the related information of nodes *F, E, B, I, F*, respectively.)

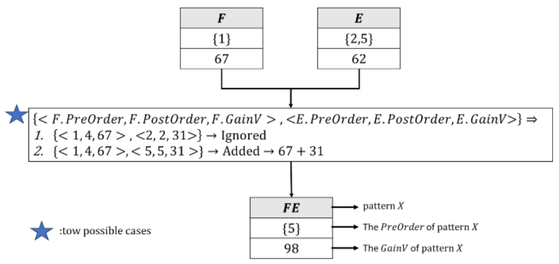
2.2.2 Pattern Expansion: After we construct the *Pattern Table* and the *Order Table*, we can expand the pattern. That is, we consider the increased size of a pattern. According to the antimonotonic property, the maximized erasable pattern which subset must be the maximized erasable pattern, so we prune the pattern which is not the maximized erasable pattern. In Figure 8, we know that items *F, E, B, I* are maximized erasable utility patterns. We could find the *MGainV* of the pattern based on the *Pattern Table* in the Figure 8 and the *Order Table* in Figure 7. For the *MEUPM_tree*, if there are two nodes in the same branch, the ancestral node contains the *GainV* of its descendant node. Then, the way to check the relation of nodes is compared to the pre-order and post-order of the nodes. If the pre-order of node A is larger than the pre-order of node *B* and the post-order of node *A* is smaller than the post-order of node *B*, node *B* must be the ancestor of node *A*. For example, in Figure 8, the first item is *F* and its preorder is 1. The second item is *E* and pre-order are 2 and 5 respectively. The pre-order of item *F* (1) is smaller than that of item *E* (2 and 5). The post-order of item *F* (4) is larger than the post-order of one node of item 2. Therefore, there are ancestor-descendant relationships between these two nodes. The *GainV* of pattern *FE* for these two nodes is the *GainV* of the ancestral node *F* (= 67). The post-order of item *F* (4) is smaller than the post-order of the other node of item 5, so there are not ancestor-descendant relationships between these two nodes. The *GainV* of pattern *FE* (98) for these two nodes is the sum of the *GainV* of node *F* (= 67) and the *GainV* of node *E* (= 31). After we compare all of the relationships between items *F* and *E*, we can know the *GainV* of pattern *FE* is 98. The *MGainV* of pattern *FE* is 294 (= 3 × 98) and it is not the maximized erasable utility pattern. According to the Theorem in the *deMERIT* + algorithm [15], if two patterns have the same prefix, we record the relative complement of pre-order. To illustrate how we efficiently consider candidates of size *k*, we use the example shown in Figure 9 for consideration. Figure 10 illustrates the overall pattern expansion process when pruning strategies are not applied, showing how all candidate patterns are generated before optimization.

2.3 Pruning Strategies:

We propose two pruning strategies through the characteristics of *MGain*. *MGain* of a pattern is calculated by multiplying *GainV* of the pattern by eu of the pattern. Because we can get the eu of the item when the data is inserted and we store them in a eu-table, we can get the eu of the pattern from table in *O(1)* time. We choose to use eu to predict the lower-bound of *MGain* of pattern efficiently.

pattern	*F*	*E*	*B*	*I*
Preorder	{1}	{2,5}	{3}	{4}

Fig. 8 The *Pattern Table* for *MEUPM_tree*



F
{1}
67
E
{2,5}
62
{< F.PreOrder, F.PostOrder, F.GainV > , <E.PreOrder, E.PostOrder, E.GainV>} ⇒
1. {< 1, 4, 67 > , <2, 2, 31>} → Ignored
2. {< 1, 4, 67 >, < 5, 5, 31 >} → Added → 67 + 31
FE
{5}
98
pattern X
The PreOrder of pattern X
The GainV of pattern X
:tow possible cases

Fig. 9 The list for the 2-length candidate pattern *FE*

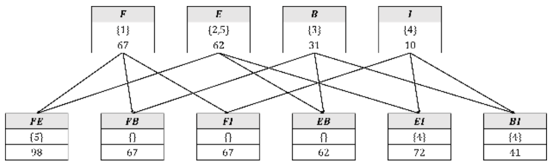


Fig. 10 The pattern expansion without using pruning strategies

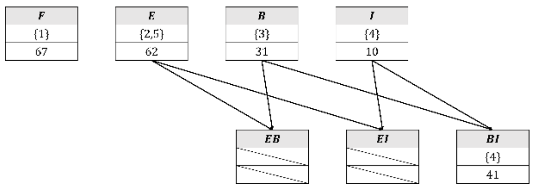


Fig. 11 The pattern expansion with pruning strategies 1 and 2

In the first pruning strategy, we can prune the pattern before the pattern expands. In the second pruning strategy, we can prune the combination of patterns when the pattern expands. The first strategy is to use the lower bound of *MGain* for the pattern before expansion.

$$LMGainV(X) = (eu(X)+minEU) \times Gain(X).$$

Note that the *minEU* is the minimum external utility of the item. Suppose there is a pattern *X* and it is an erasable pattern. That is, *MGain(X)* could be smaller than or equal to the *MGT*. The minimum value of the external utility in the database is 1. Assume there is any item *Y* and it is also erasable. Because the real *eu(XY)* is the sum of external utilities of items *X* and *Y*, the *eu(X) + minEU* could be absolutely smaller than or equal to the *eu(XY)*. Because the property of *GainV*, *GainV (XY)* must be larger than or equal to the *GainV(X)* The real *MGain(XY)* must be larger than the *LMGain(X)* for any item *Y*. If the *LMGain(X)* is larger than *MGT*, it means the combination of pattern *X* with others is not an erasable pattern. Then, we can output pattern *X* and prune pattern *X* before expansion. For example, if *MGT* is 173.5 and *minEU* is 1, the maximized erasable utility items are *{F, E, B, I}*. The *GainV* of item *F* is 67 and the *eu* of item *F* is 2. *LMGainV(F)* is 201(= (2+1)×67). Because the *LMGainV* of item *F* is larger than *MGT*, the combinations of item *F* (*i.e.* pattern *FE,FB,FI*) are non-erasable. We can save time in pattern *F* expansion. The second strategy is to make use of the lower bound of *MGain* for the pattern in the expansion stage.

$$PLMGainV(XY) = (eu(X) + eu(Y)) \times max(GainV(X), GainV(Y))$$

Note that the *MAX(GainV(X), GainV(Y))* is the maximum of *GainV* of patterns *X* and Y. Suppose there are two erasable patterns *X* and *Y*. Before we combine patterns *X* and *Y*, we have already known the *GainV* of patterns *X* and *Y*. *MGain(XY)* is calculated by multiplying *GainV* of pattern *XY* by the *eu* of pattern *XY*. Note that the *eu* of pattern *XY* is calculated by the sum of the *eu* of patterns *X* and *Y*.

Because of the definition of *GainV*, the *GainV(XY)* must be larger than or equal to the *max(GainV(X), GainV(Y))*. The real *MGain(XY)* must be larger than or equal to *PLMGainV(XY)*. If the *PLMGainV(XY)* is larger than *MGT*, it means that *MGain(XY)* is higher than *MGT*. Then, we can prune the pattern before calculating the *GainV(XY)*.

For example, if *MGT* is 173.5, the maximized erasable utility items are *{F, E, B, I}*. The *GainV* of item *B* is 31 and the *eu* of item *B* is 2.The *GainV* of item *E* is 62 and the *eu* of item *E* is 1. *PLMGainV(BE)* is 186 (= (2+1) × 62). Because the *PLMGainV* of pattern *BE* is higher than *MGT*, the pattern *BE* is non-erasable. We can save the time of looking for the pattern *BE* combination. Figure 11 shows the pattern expansion using pruning strategies 1 and 2. (Note that in Figure 11, in fact, originally, there are three nodes *FE*, *FB*, *FI* and six edges: *F*→*FE*, *F*→*FB*, *F*→*FI*, *E*→*FE*, *B*→*FB*, *I*→*FI*. The reason is that the Gain value of node *F* is larger than the threshold.)

2.4 *The Mining Process for the Incremental Database*

After we get the maximized erasable utility patterns from the original database, in this section, we will consider the mining method for the incremental database. First, we will clear the *MEUPM*_tree, the *Order Table*, and the *Pattern Table*. We will initialize other parameters. When the new transaction is inserted, we calculate the profit of the transaction and update the *PT*. If the new transaction contains an item that has never appeared before, we need to re-scan the external utility database. After we calculate the profit, we will create nodes of items in the transaction. Next, we insert nodes into the *EUP_tree* and update the *Header Table*.

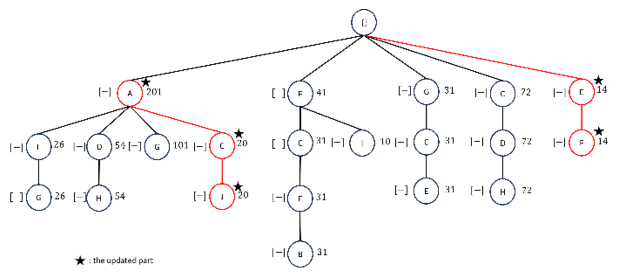


Fig. 12 The *EUP_tree* after updated

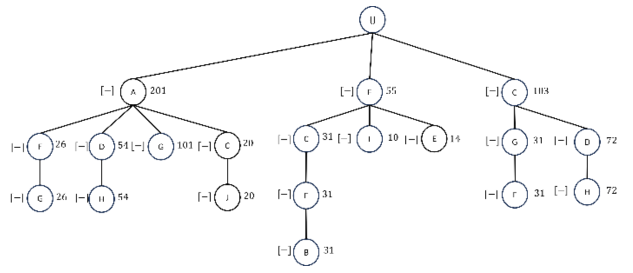


Fig. 13 The *EUP_tree* after reconstructed

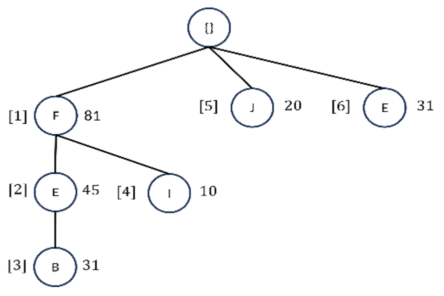


Fig. 14 The *MEUPM_tree* after doing the preorder traversal

For example, let's consider the input set T_8 = [*E(4), F(5)*] and T_9 = *[A(2), C(3), J(1)]*. Item *J* in T_{10} is an item that has never appeared before. To update the *eu* table, we need to re-scan the external utility database. The external utilities of items *A* through *J* are 4, 2, 3, 6, 1, 2, 3, 4, 2, and 3, respectively. Then, we create the nodes of items in new transactions and update the *PT*. After the insertion of new transactions, the *PT* values for T_1 through T_9 are 26, 31, 54, 31, 72, 10, 101, 14, and 20, respectively. We insert nodes into the *EUP_tree* and update the *Header Table*. The *Header Table* after new transactions are inserted shows the updated values. Values of *Gain(A), Gain(F), Gain(C), Gain(G), Gain(E), Gain(D), Gain(H), Gain(B), Gain(I)* and *Gain(J)* are 201, 81, 154, 158, 76, 126, 126, 31, 10, and 20, respectively, with corresponding support values of 4, 4, 4, 3, 3, 2, 2, 1, 1, and 1, respectively. Figure 12 shows the *EUP_tree* after new transactions are inserted.

After all transactions are inserted, we calculate the *MGT*. For example, if the threshold δ is 0.7, the *MGT* is 251.3. Next, we reconstruct the *EUP_tree* and construct the *MEUPM*_tree. Figure 13 shows the *EUP_tree* after reconstruction.

Next, our approach is the same as the static method. We will generate the new *Pattern Table* and the new *Order Table* by the *MEUPM_tree*. We do the preorder traversal to set the *PreOrder* of nodes in the *MEUPM* tree and generate the *Pattern Table*. Figure 14 shows the new *MEUPM_tree* after doing the preorder traversal. The new *Pattern Table* after performing the preorder traversal lists each pattern alongside its corresponding preorder number: *F* is {1}, *E* is {2, 6}, *B* is {3}, *I* is {4}, and *J* is {5}. Figure 15 shows the new *Order Table* after doing the postorder traversal. (Note that in Figure 15, it stores the related information of nodes *F, E, B, I, J, F,* respectively.) Then, we mine erasable utility patterns with pruning strategies by the *Pattern Table* and the *Order Table*.

Preorder	1	2	3	4	5	6
Postorder	4	2	1	3	5	6
GainV	67	31	31	10	20	31

Fig. 15 The *Order Table* after doing the postorder traversal

3 Performance

In this section, we will present the performance study of our proposed *MEUPT* algorithm and the *MEL* algorithm [6] for mining maximized erasable utility patterns under several situations. Databases are real-life transaction datasets with synthetic (fake) utility values. We will introduce the performance model [16]. The first parameter, *Eth,* means the maximized erasable utility pattern threshold. The value of *Eth* is the percentage. The second parameter, *NT*, means the number of transactions in the *TDB*. The third parameter, *NI*, means the number of distinct items in the *TDB*. The fourth parameter, *MI*, implies the mean number of items in a single transaction.We use two real datasets in our experiments: connect (sparse) and chess (dense), downloaded from the *SPMF* repository (https://www.philippe-fournier-viger.com/spmf/datasets. php). Figure 16 shows the real-life transaction datasets. External utilities are randomly assigned. Density is computed as (*MI*/*NI*)×100%. To simulate incremental mining, each dataset is divided into two parts. Since *MEL* is static, it reprocesses the entire database, whereas *MEUPT* updates patterns incrementally. For the connect dataset (33.33% density), Figure 17(a) shows that runtime is similar at low *Eth* due to the small number of patterns and the cost of maintaining transaction information. As *Eth* increases, *MEUPT* becomes significantly faster because its tree structure merges item information effectively and enables stronger pruning. In the incremental case (Figure 17(b)), *MEUPT* consistently outperforms *MEL*, which must restart from scratch. For the dense chess dataset (49.33% density), both static and incremental results (Figures 17(c)–(d)) show the same trend: MEUPT achieves superior performance due to efficient data merging and more effective pruning strategies.

Dataset	NT	NI	MI	Density(%)
connect	67577	129	43	33.33
chess	3196	75	37	49.33

Fig.16 Real transaction datasets

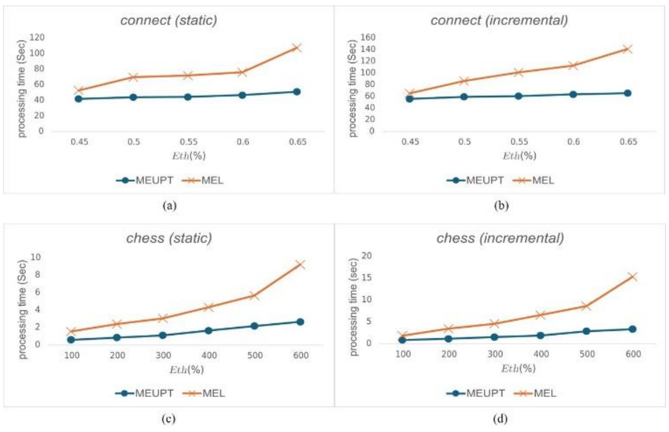


Fig. 17 Performance Comparison of real datasets (Connect and Chess) between *MEUPM* and *MEL*: (a) Connect (static); (b) Connect (incremental); (c) Chess (static); (d) Chess (incremental).

4 Conclusion

In this paper, we have proposed the *MEUPT* algorithm, which is able to mine the maximized erasable utility patterns efficiently. The proposed *MEUPT* algorithm only scans the database once and sorts database once. We have proposed efficient pruning strategies to delete candidates early. Moreover, we have considered the incremental database. From our performance evaluation using real datasets, we have shown that our algorithm is more efficient than the *MEL* algorithm.

5 Acknowledgements

This work was supported by the National Science and Technology Council (NSTC) of Taiwan under Grant No. 114-2221-E-110-022. The authors gratefully acknowledge the Council's support, which made it possible to undertake this research and participate in the related academic conference.

6 References

[1] W. Kai, X. Meng-meng, and Z. Xu-Hong, “A Weighted Erasable Itemset Mining Algorithm Based on List Structure,” Computer Engineering and Science, vol. 43, pp. 1676–1683, September 2021.
[2] G. Lee and U. Yun, “Single-pass Based Efficient Erasable Pattern Mining Using List Data Structure on Dynamic Incremental Databases,” Future Generation Computer Systems, vol. 80, pp. 12–28, July 2018.
[3] H. Nam, U. Yun, E. Yoon, and J. C.-W. Lin, “Efficient Approach for Incremental Weighted Erasable Pattern Mining with List Structure,” Expert Systems with Applications, vol. 143, pp. 113087–113109, November 2020.
[4] U. Yun, G. Lee, and E. Yoon, “Advanced Approach of Sliding Window Based Erasable Pattern Mining with List Structure of Industrial Fields,” Information Sciences, vol. 494, pp. 37–59, April 2019.
[5] T. Le, B. Vo, P. F. Viger, M. Y. Lee, and S. W. Baik, “SPPC: A New Tree Structure for Mining Erasable Patterns in Data Streams,” Applied Intelligence, vol. 49, pp. 478–495, February 2019.
[6] C. Lee, Y. Baek, T. Ryu, H. Kim, H. Kim, J. C.-W. Lin, B. Vo, and U. Yun, “An Efficient Approach for Mining Maximized Erasable Utility Patterns,” Information Sciences, vol. 609, pp. 1288–1308, July 2022.
[7] G. Lee, U. Yun, H. Ryang, and D. Kim, “Erasable Itemset Mining over Incremental Databases with Weight Conditions,” Engineering Applications of Artificial Intelligence, vol. 52, pp. 213–234, June 2016.
[8] B. Vo, T. Le, G. Nguyen, and T.-P. Hong, “Efficient Algorithms for Mining Erasable Closed Patterns from Product Datasets,” IEEE Access, vol. 5, pp. 3111–3120, March 2017.
[9] H. Nguyen and T. Le, “A Fast Algorithm for Mining Top-rank-k Erasable Closed Patterns,” Computers, Materials and Continua, vol. 72, pp. 3571–3583, March 2022.
[10] G. Nguyen, T. Le, B. Vo, and B. Le, “Discovering Erasable Closed Pattern,” Proc. of Asian Conference on Intelligent Information and Database Systems, vol. 9011, pp. 368–376, January 2015.
[11] Y. Baek, U. Yun, J. C.-W. Lin, E. Yoon, and H. Fujita, “Efficiently Mining Erasable Stream Patterns for Intelligent Systems over Uncertain Data,” International Journal of Intelligent Systems, vol. 35, pp. 16991734, July 2020.
[12] C. Lee, Y. Baek, J. C.-W. Lin, T. Truong, and U. Yun, “Advanced Uncertainty Based Approach for Discovering Erasable Product Patterns,” Knowledge-Based Systems, vol. 241, pp. 108134–108152, January 2022.
[13] T.-P. Hong, K.-Y. Lin, C.-W. Lin, and B. Vo, “An Incremental Mining Algorithm for Erasable Itemsets,” Proc. of IEEE International Symposium on Innovations in Intelligent Systems and Applications, pp. 286289, July 2017.
[14] L. Nguyen, G. Nguyen, and B. Le, “Fast Algorithm for Mining Maximal Erasable Patterns,” Expert Systems with Applications, vol. 124, pp. 5066, January 2019.
[15] B. Vo, T. Le, and F. Coenen, “An Efficient Algorithm for Mining Erasable Itemsets Using the Difference of NC-sets,” Proc. of IEEE International Conference on Systems, Man, and Cybernetics, pp. 22702274, October 2013.
[16] P. Fournier-Viger, C. W. Lin, A. Gomariz, T. Gueniche, Z. D. A. Soltani, and H. T. Lam, “The Spmf Open-Source Data Mining Library Version 2,” Proc. 19th European Conf. on Principles of Data Mining and Knowledge Discovery (PKDD 2016) Part III, pp. 36–40, 2016.