A SURVEY OF ERASABLE ITEMSET MINING ALGORITHMS FOR DYNAMIC INCREMENTAL DATABASE

***Linh Nguyen*[1]**
[1]*Faculty of Information Technology, Ho Chi Minh City University of Economics and Finance*
linhnth@uef.edu.vn

Abstract
Erasable itemset mining is one of the interesting areas in frequent itemset mining. Analyzing incremental data becomes more important because interesting data are continually accumulated in various application fields including industrial areas. This article provides a survey of the available erasable itemset (EI) mining on dynamic incremental databases. Incremental mining algorithm for erasable itemsets was first presented in 2016 and IWEI is the first algorithm for discovering erasable itemsets considering weight conditions in incremental databases. Since then, a number of algorithms, such as FUP-erasable, LINE and WEIL have been proposed mining EI on incremental data. In this study, the IWEI, FUP-erasable, LINE and WEIL algorithms are described and compared in terms of mining time and memory usage.
Key words*: Data mining, Erasable itemsets, Incremental mining, Weighted Erasable itemsets*

1. Introduction

In 2009, Deng et al. defined the problem of erasable itemset (EI mining), which is a variant of itemsets mining [3] for mining product database.In recent years, several algorithms have been proposed for EI mining, such as META (Mining Erasable iTemsets with the Anti-monotone property) [3], VME (Vertical-format-based algorithm for Mining Erasable itemsets) [4], MERIT (Fast Mining Erasable ITemsets) [5], dMERIT+ (using difference of NC_Set to enhance MERIT algorithm) [6], and MEI (Mining Erasable Itemsets) [7].

Although previous algorithms discover erasable itemsets, their coverages are limited to static product databases only. But products database can be added continually and frequently. Since traditional methods have to perform their own mining processes from scratch even if a small change occurs in the accumulated data, they are inefficient in processing such data. In 2016, Lee et al. defined the problem of EI mining for dynamic incremental databases with IWEI [8] algorithm. In recent years, several algorithms have been proposed for EI mining on dynamic incremental databases, such as FUP-erasable [9], LINE [10] and WEIL [11].

This study outlines existing algorithms for mining EIs on incremental databases. For each algorithm, its approach is described, an illustrative example is given, and its advantages and disadvantages are discussed.

The rest of this study is organized as follows: Section 2 introduces the theoretical basis of EI mining on static and dynamic databases; Section 3 presents IWEI, FUP-ERASABLE, LINE and WEIL algorithms; Section 4 gives the conclusion and suggestions for future work.

2. Related work

2.1. Erasable Itemset mining for static databases

The concept of erasable itemset mining was proposed to solve financial crises that may occur in manufacturing plants. Given a product database, each transaction signifies a production line and items of the transaction become components composing the corresponding product. Then, an erasable itemset means a set of components to be excluded from previous production lines, where each erasable itemset has a total profit lower than or equal to a user-specified threshold. Currently, there are numerous algorithms that are effectively mining erasable itemsets. In an initial erasable itemset mining approach based on a level-wise method with data structures are mainly categorized as two types:

1. List structures: List based approaches store product information including IDs and profits of products into list structures and

then utilize this information during their own mining processes. META [3], VME [4] and MEI [7] are three such algorithms.

2. Tree structures: tree-based methods construct tree structures from product databases and then conduct their mining works using the constructed trees without any further database scan. MERIT [5], MERIT + [6] and dMERI + [6] are three such algorithms.

2.2. Erasable Itemset mining for dynamic incremental databases

Let I ={i1, i2,…, im} be a set of all items, which are the abstract representations of components of products. A product dataset, PDB, with n product transactions, PID, { $P_1, P_2, \ldots, P_n$}. Each product P_i is represented in the form *⟨Items, Val⟩*, where Items are all items that constitute P_i and Val is the profit that the factory obtains by selling product P_i. A set X ⊆I is called an itemset, and an itemset with k items is called a k-itemset. Each product P_i is represented in Table 1.

Table 1. *Example of a product database for erasable itemset mining*

PID	Items	val ($)	
P_1	a d e	1100	PDB_0
P_2	a c d e	300	
P_3	c d g	500	
P_4	b d	400	
P_5	e g h	600	
P_6	a g	200	
P_7	a b g	700	PDB_1
P_8	b d e	500	
P_9	c f	300	
P_{10}	b d g h	600	PDB_2
P_{11}	g h i	500	
P_{12}	e f	400	

The general definition of a product database is as follows.

Definition 1. Let X = {$i_1, i_2, i_3 \ldots, i_k$} i an itemset generated from PDB the gain of X is defined as:

$$\boldsymbol{Gain(X)} = \sum_{\{P_k | X \cap P_k.itemset \neq \emptyset\}} P_k.va \quad (1)$$

This means that the gain in X be computed by summing all vals of products.

Definition 2. (**Maximum gain threshold**) A threshold value is used to check validity of each found itemset, which is called Maximum Gain Threshold (MGT) and calculated as follows:

$$MGT = \xi \times \sum_{P_k \in PDB} P_k.val \quad (2)$$

In Equation (2), P_k is a product transaction in PDB and ξ is a threshold percent value given by user.

Definition 3. (Erasable itemset) Let X = {i_1, i_2, $i_3 \ldots$, i_k} be an itemset composed of distinct items, If X has:

$$Gain(X) \leq MGT \quad (3)$$

It is regarded as an erasable itemset EI; otherwise, it becomes a non-erasable one.

Let PDB_0 be an original database with the first 6 product transactions of the database in Table 1, PDB_1 be the first incremental database with the next 3 transactions, and PDB_2 be second one with the remainders.

3. Existing algorithms for EI mining for incremental databases

This section introduces existing algorithms for EI mining for incremental databases, namely IWEI, FUP-ERASABLE, LINE, and WEIL.

IWEI Algorithm

Algorithm

In 2016, Lee et al. has proposed an algorithm IWEI for discovering erasable itemsets considering weight conditions in incremental databases [8]. IWEI uses a tree-based weighted erasable itemset mining approach devised to deal with incremental data. IWEI use a technique that utilizes weight conditions considering the characteristics of erasable itemset mining. This algorithm is mainly divided into two phases: tree updates and weighted erasable itemset mining. The basic concepts associated with this structure are as follows.

This algorithm additionally considers weights of items shown in Table 2 and incremental databases like in Table 1.

Table 2. *Weight table*

Item	a	b	c	d	e	f	g	h	i
Weight	0.4	1.0	0.6	0.4	0.2	0.8	0.8	0.4	0.6

Definition 4. (IWEI-tree) An IWEI-tree is composed of a header table and a prefix tree. The header table has three columns

<item name, frequency (F), profit>

The prefix tree consists of multiple nodes, where each node N includes:

<N.item-name, N.frequency, N.total-val, N.parent-link, N.children-set, N.pre-order-index, N.res-flag>

Definition 5. Order-and-profit-list (**OPlist**). OP-list is generated after the IWEI-tree construction, where the list consists of pre-order index, total-profit, and post-order index information of the nodes composing the tree. Each element of OP-list includes information corresponding to each node of the tree.

Definition 6. Let PDB be a given product database and X = {i_1, i_2, ..., i_s} be a s-itemset (s ≥ 1). Then, an **average weight** of X, AW(X), is calculated as follows:

$$AW(X) = \frac{\sum_{p=0}^{S} AW(i_p)}{S} \quad (4)$$

Let P(i_p)={P_1, P_2, ..., P_k} be a set of products including i_p. Then, AW(i_p) is defined as the following equation:

$$AW(i_p) = \frac{\sum_{j=1}^{k} \left(\frac{w(i_p)}{\sum_{i_r \in P_j} w(i_r)} \right)}{k} \quad (5)$$

Definition 7. A weighted gain of X, WGain(X), as follows:

$$WGain(X) = \frac{Gain(X)}{AW(X)} \quad (6)$$

if an itemset, X, satisfies the following condition, X is regarded as **a weighted erasable itemset**.

$$WGAIN(X) \leq MGT$$

Definition 8. A maximum weighted gain of an itemset, X, MGain(X), is calculated as follows:

$$MGain(X) = \frac{Gain(X)}{MAW} \quad (7)$$

Where MAW is the largest AW value among the AW values of itemsets with length 1

The details of IWEI are given in Fig. 1.

Discussion

IWEI algorithm constructs IWEI-tree tree structure, within a single database scan in order to process dynamic incremental data. Although IWEI is able to perform erasable itemset mining works from incremental product databases, it still has performance limitations for the following reasons:

1. Tree constructing and restructuring be complicated, lead to high overheads.
2. Low utilization of the tree structure compared to its high maintenance cost.

Input: an original product database, $\mathbf{PDB_0}$, an incremental product database, $\mathbf{PDB_1}$, the weight information for each item, **W**, and a maximum gain threshold, **MGT**
Output: a set of weight erasable itemsets **WEI**
Variables: a *IWEI-tree, **Tree***, an *OP-list, **OPList***, a set of *AW* of 1-itemsets, $\boldsymbol{AW_1}$, candidates old weighted erasable 1-itemset, ***CI***, and the Boolean value for the restructure flag, ***RF***
If *RF* is not initialized, **Then** *RF* ← false
For each product, ***P***, in PDB_0 or PDB_1, **do**
    N* ← *the root node of Tree
    Sort items in ***P*** in accordance with the current sorting order
    Calculate a total weight $\boldsymbol{W_P}$ of all items in ***P***
    For each item *i* in *P,* **do**
        If *N.children-set* = ∅, *CN*, where *CN.item-name* = *I*, **Then**
            N.children-set* ← N_C, N_C*.item-name* ← *I*, N_C*.frequency* ← *1

```
            N_C.total-val ← P.Val, N_C.parent-link ← N, N_C.children-set ← ∅
            N_C.res-flag ← RF, N ← N_C
        End if
        Update the information of i in AW_1 by using W_P
    End for
MAW ← the maximum AW value in AW_1, RF ← ¬RF, N_r ← the root node of Tree
Call Restructure_Tree(N_r, RF)
CI ← ∅ and WEI ← ∅
For each item, i_1, in the whole product database, do
    If (Gain(i_1)/MAW) <= MGT, Then
        CI ← i_1
        If (Gain(i_1)/ AW_1(i_1)) <= MGT, Then WEI ← i_1
    End if
End for
Generate OPList and Input pre-order-index for each node in Tree
Call Mine_WEI(Tree, CI, AW_1, OPList, WEI)
```

```
Procedure Restructure_Tree(N_r, RF)
Variables: temporary array for sorting a path, R_t
For each child node N_rc of N_r, do
    Call Restructure_Tree(N_r, RF)
    If N_rc.children-set = ∅, Then
        R_t ← ∅, N_t ← N_rc, COUNT ← N_t.frequency, VAL ← N_t.total-val
        For each N_t, do
            R_t ← N_t, N_t.frequency ← N_t.frequency – COUNT
            N_t.total-val ← N_t.total-val – VAL, N_t ← N_t.parent-link
        End for
    End if
    Sort R_t according to the current sorting order
    Insert the information of R_t into Tree by using RF
    If N_rc.frequency = 0, Then Delete N_rc
End for
```

```
Procedure Mine_WEI(Tree, CI, AW, OPList, WEI)
Variables: a set of weighted candidate itemsets, CI_t, a weighted candidate itemset, C_i
CI_t ← CI , CI ← ∅
For each itemset IT_i in CI_t, do
    For each itemset IT_j in CI, do        // IT_j != IT_i
        Generate a new candidate itemset, C_i by using IT_i and IT_j
        If (Gain(C_i)/MAW) <= MGT, Then
            CI ← C_i
            If (Gain(C_i)/ AW(C_i)) <= MGT, Then WEI ← C_i
    End for
End for
If |CI| >= 2, Then Call Mine_WEI(Tree, CI, AW, OPList, WEI)
```

Figure 1. *EWEI algorithm*

FUP-ERASABLE Algorithm

Algorithm

Hong et al. proposed the FUP algorithm to mining erasable itemsets from newly inserted transactions [9]. Similar to FUP for association rules [1], this algorithm is divided into four cases of incremental erasable itemset mining:

- Case 1: an itemset is erasable in the original database and no items or erasable in the newly inserted transactions.
- Case 2: An itemset is erasable in the original database but is non-erasable in the newly inserted transactions.
- Case 3: An itemset is non-erasable in the original database but is no items or erasable in the newly inserted transactions.
- Case 4: An itemset is non-erasable in both the original database and the newly inserted transactions.

The itemsets in Case 1 are divided into two parts to be processed. The first part includes the itemsets which are erasable in the original product database but with no items in the new products. They will still be erasable after the process. The second part includes the itemsets which are erasable in both the original product database and the new product database. They will still be erasable after the process. Similarly, itemsets in Case 4 will still be non-erasable after the new product database is inserted. Thus Cases 1 and 4 will not affect the final erasable itemsets. Case 2 may remove existing erasable itemsets, and Case 3 may add new erasable itemsets.

The details of FUP-ERASABLE are given in Fig. 2.

Discussion

Among four cases, cases 1 and 4 will not affect the final erasable itemsets. Case 2 may remove existing erasable itemsets, and case 3 may add new erasable itemsets. Based on the FUP approach, cases 1, 2 and 4 are more efficiently handled than conventional batch mining algorithms. It much however re-process the original database for managing case 3. Incremental approaches have to guarantee not only correctness of itemset mining results but also good algorithm performance because they should immediately process such large-scale data and provide users with results itemset mining. However, FUP method has difficulty in satisfying these conditions.

Input: An original product database P with its total profit value T^P, the set of erasable itemsets E^P with their gain value from P, an erasable ratio threshord r, the set of all items I, and a set of newly coming products N.

Output: The set of erasable itemsets for the updated database U.

Variables:

T^N: The total profit value of N

C_1^N: List the 1-itemsets appearing in N

C_1^P: The candidate 1-itemsets not appearing in N

C_k^N: The set of candidate erasable k-itemsets from N

E_k^N: The set of erasable k-itemsets from N

E_k^P: The set of erasable k-itemsets from P

E_k^U: The set of updated erasable k-itemsets

$gain^U(s)$, $gain^P(s)$ and $gain^N(s)$: The gain of s in U, P and N, respectively.

Step 1: Calculate T^N , as follows: $T^N = \sum_{p_i \epsilon N} p_i.value$

Step 2: Generates C_1^N and calculate their gain values

Step 3: Set $C_1^P = I - C_1^N$

Step 4: Set k = 1

Step 5: For each k – itemset s in C_k^N, **do**

If $gain^N s \leq T^N \times r$ **Then** $E_k^N \leftarrow C_k^N$

Step 6: For each k – itemset s in E_k^P, **do**
If s ∈ E_k^N **Then** $gain^U(s) = gain^P(s) + gain^N(s)$, $E_k^U \leftarrow$ s
Step 7: For each k – itemset s in E_k^P and also in C_k^N but not in E_k^N , **do**
$gain^U(s) = gain^P(s) + gain^N(s)$
If $gain^U s \leq (T^P + T^N) \times r$ **Then** $E_k^U \leftarrow$ s
Step 8: For each k – itemset s in E_k^P, **do** $E_k^U \leftarrow$ s
Step 9: For each k – itemset s which exists in (E_k^N-E_k^P) or in C_k^P-E_k^P, **do**
Rescan P to determine $gain^P(s)$
If s is in (E_k^N-E_k^P) **Then** $gain^U(s) = gain^P(s) + gain^N(s)$
Else $gain^U(s) = gain^P(s)$
If $gain^U s \leq (T^P + T^N) \times r$ **Then** $E_k^U \leftarrow$ s
Else remove s from E_k^N
Step 10: Generates (k+1)-itemsets C_{k+1}^U from k-itemsets in E_k^U.
For each (k+1)-itemset s in C_{k+1}^U, **do**
If s includes at least one item in the new products, **Then**
$C_{k+1}^N \leftarrow$ s, Calculate $gain^N(s)$
Else $C_{k+1}^P \leftarrow$ s
Step 11: Set k = k+1
Step 12: Repeat Steps 5-11 until no new candidate erasable itemsets are generated.
Step 13: Output all the itemsets in E_k^U

Figure 2. *FUP-ERASABLE algorithm*

LINE algorithm

Algorithm

Lee et al. proposed the LINE algorithm incremental erasable itemset mining [10]. Suggesting new itemset pruning and mining techniques that can more effectively perform incremental erasable itemset mining operations though the proposed data structures.

The basic concepts associated with this structure are as follows.

Definition 10. LINE-table. LINE-table is a table structure composed of indexes and vals of transactions belonging to the database.

PID	P_1	P_2	…	P_k
Val	P_1.Val	P_2.Val	…	P_k.Val

Figure 3.*General architecture of LINE-table*

Definition 11. EI-list. EI-list is a data structure for storing information refined from a given product database

Item	i_k
PIDs	$P_1, P_2, …, P_k$
Gain	Gain(i_k)

Figure 4.*General architecture of EI-list*

Definition 12. EP-list. EP-list is generated from the constructed EI-list. One EP-list is constructed for each itemset. Each EP-list store:

i. Information of each mine itemset with two or longer length
ii. A difference set of PIDs instead of storing all the PID indexes.

Itemset	i_k
dPIDs	$P_1, P_2, …, P_k$
Gain	Gain(i_k)

Figure 5. *General architecture of EP-list*

Definition 13. PID range index (PRI). Given $S_1 = \{idx_1, idx_2, \ldots, idx_m\}$ and $S_2 = \{idx'_1, idx'_2, \ldots, idx'_n\}$ is two PID or dPID sets. PRIs of S_1 and S_2 can be denoted as follows:

If $idx_1 < idx'_1$, **Then**
If $idx_m < idx'_1$, **Then** *Construct EP-list in the intersection-based manner*
Else *Construct EP-list in the difference-based mannner*
Else

If $idx'_n < idx_1$, **Then** *Construct EP-list in the intersection-based manner*
Else *Construct EP-list in the difference-based manner*
End if

Figure 6. *EI-list construction using the features of the intersection between PID sets*

Definition 14. Lower bound in EP-list (LB). Let $Val_{min}(PDB)$ be the smallest val value in an incremental product database, PDB. Then, given two erasable itemsets, X and Y, the lower bound of Gain(XY), LB(XY), is calculated as follows:

LB(XY) = Gain(X) +(Val_{min}(PDB)×# of items in difference set of PID sets of X·Y

Definition 15. Permanently inerasable itemset. If LB(XY) is larger than MGT, XY becomes a permanently inerasable itemsets, and all of the works related to XY are omitted. Let XY be an itemsets with n2 or longer length than can be combined from two erasable itemstes, Then, its always true that LB(CY) ≤ Gain(XY)

The details of LINE are given in Fig. 10, Fig.11 and Fig. 12.

LINE would accumulate new data (original data at first, after that, incremental data). After that, algorithm construct or update LINE-table and EI-lists from the inputted data. After EI-list and LINE-table are constructed from a given product database, the algorithm waits for a mining request by the user or input of new incremental data. If a mining request occurs, the algorithm recursively generates EP-list from the constructed EI-list. If a new incremental data is inputted, the algorithm repeats update LINE-table and EI-list.

Discussion

EI-list and LINE-table that are constructed only once and continually update during the entries mining process. Although the algorithm has applied the lower bound and PRI for pre-pruning of inerasable itemset, whenever the user sends a mining request, LINE proceeded to mine from the beginning. It also takes a lot of memory and time because the new data is small.

WEIL algorithm

Algorithm

Nam et al. proposed WEIL algorithm mines weighted erasable itemsets in an incremental environment [11]. The algorithm uses WEI-list and WEP-list to store the weight values of items and given database. A detailed of data structures used are as follows.

Definition 16. WEIL-table. WEIL-table is a table structure composed of indexes and vals of transactions belonging to the database.

PID	**P_1**	**P_2**	**…**	**P_k**
Val	**P_1.Val**	**P_2.Val**	**…**	**P_k.Val**

Figure 7. *General architecture of WEIL-table*

Definition 17. WEI-list. EI-list is a data structure for storing information refined from a given product database

Item	**i_k**
PIDs	**P_1, P_2, …, P_k**
Gain	**Gain(i_k)**

Figure 8. *General architecture of WEI-list*

Definition 18. AW-table. AW-table is a data structure for storing the average weight value of each item. Each time a new transaction is inserted, the value of the corresponding item in the AW-table is updated.

Item	**i_1**	**i_2**	**…**	**i_k**
AW	**AW(i_1)**	**AW(i_2)**	**…**	**AW(i_k)**

Figure 9. *General architecture of AW-table*

Definition 19. WEP-list. WEP-list is generated from the constructed EI-list when a mining request is received. The WEP-list stores the difference set of PIDs and gain value of the item.

Itemset	**i_k**
dPIDs	**P_1, P_2, …, P_k**
Gain	**Gain(i_k)**

Figure 10. *General architecture of WEP-list*

The WEP-list is deleted when the itemset expansion process is finished.

The details of WEIL are given in Fig. 14.

First, WEIL constructs an initial WEI-list by scanning a given original database and

LINE
Input: an original product database, $\mathbf{PDB_0}$, a set of incremental product databases, **IncDB= {** $\mathbf{PDB_1, PDB_2, \ldots}$**}**, and a maximum gain threshold, **MGT**
Output: a set of t erasable itemsets, **R**
Variables: a *Line-table*, ***LT,*** a set of *EI-list,* ***EI***
LT = ∅ and **EI** = ∅
Call ***Construct_PDB(LT, EI,*** $\boldsymbol{PDB_0}$***)***
If a mining request occurs, **Then**
    Calculate ***MGT***
    For earch item s in EI, **do If** *Gain(s)* ≤ *MGT* **Then** *EI'* ← *s*
    R = R ∪ EI'
    Call ***Mine_Itemsets(LT, EI', MGT,R)***
    Provide the user with ***R***, ***R* = ∅**
End if
For earch incremental product, PDB_i, in *IncDB,* **do**
    Call ***Construct_PDB(LT, EI,*** $\boldsymbol{PDB_i}$***)***
    If a mining request occurs, **Then**
        For earch item s in EI, **do If** *Gain(s)* ≤ *MGT* **Then** *EI'* ← *s*
        R = R ∪ EI'
        Calculate ***MGT***
        Call ***Mine_Itemsets(LT, EI, MGT,R)***
        Provide the user with ***R***, ***R* = ∅**
    End if
End for

Figure11. *Main procedure of LINE*

Construct_PDB
Input: A *LINE-table*, **LT**, a set of *EI-list*, **EI**, a target product database, **TPDB**
Output: LT and **EI** update with TPDB
For each product P_k in *TPDB*, **do**
    For earch a new element, ***e***, **do** $e.PIDs = P_k.PID,\ Gain(e) = P_k.Val,\ LT \leftarrow e$
    For earch item i_k in P_k, **do**
        If *ei* of i_k already exists EI, **Then**
            $ei.PIDs = ei.PIDs \cup P_k.PID,\ Gain(ei) = Gain(ei) + P_k.Val$
        Else
            EI ← ei' ,with ei' be a new EI-list for i_k
            ei'.PIDs = $\boldsymbol{P_k.PID}$***, Gain(ei') =*** $\boldsymbol{P_k.Val}$
        End if
    End for
End for

Figure 12. *Sub_procedure Construct_PDB of LINE*

Mine_Itemsets
Input: A LINE-table **LT**, a set of EI-list, **EI**, or a set of EP-list, **EP**, a maximum gain threshold, **MGT**, a set of erasable itemsets, **R**
Out put: R
***EP'* = ∅**, EP' be a EP-list
For each item *Ii* in EI or EP, **do**

For each item I_j in EI or EP, **do** //i<j,
  ep'.Itemset = I_i.Item ∪ I_j.Item // I_i.Itemset ∩ I_j.Itemset in EP
  If I_i and I_j satisfy the PRI condition, **Then**
    ep'.dPID = I_j.PIDs or I_j.dPIDs, Gain(ep') = Gain(I_i) + Gain(I_j)
    If Gain(ep') ≤ MGT, **Then *EP' = EP' ∪ ep', R = R ∪ ep'***
  Else
    If LB(ep') ≤ MGT, **Then**
      ep'.dPID = I_i.PIDs·I_j.PIDs or I_i.dPIDs·I_j.dPIDs
      Gain(ep') = Gain(I_i) + Gain(ep'.dPIDs)
      If Gain(ep') ≤ MGT, **Then *EP' = EP' ∪ ep', R = R ∪ ep'***
    End if
  End if
End for
If count(EP') > 1, **Then** Call **Mine_Itemsets(LT, EP', MGT, R)**
End for

Figure 13. *Sub_procedure Mine_Itemsets of LINE*

computes an average weight for each item in the database with the weight information of its. After that, if a new incremental data is inputted, new data information is added to the constructed WEI-list and the average weight values are updated. If a mining request occurs, the algorithm is computes MGT use WEI-list. Then, the algorithm finds 1-length weighted erasable candidate itemset by calculating MGain and only itemsets have Wgain less than or equal to the MGT insert into WEI-list. And then, the operation of extending the itemset is recursively repeated until no new candidate is created. The algorithm recursively extracts weighted erasable itemsets with their WGain value is less than or equal to the MGT (with only apply itemsets have been MGain value is less than or equal to the MGT).

Discussion

By using the list data structure, WEIL algorithm can scan a dynamic incremental database which is gradually increasing only once, stores data efficiently and performs mining operations. This algorithm shows better performance than the previous techniques in terms of run time, memory usage, and scalability. However, it still has performance limitation same LINE algorithm that is whenever the user sends a mining request, WEIL proceeded to mine from the beginning based on WEI-list, WEIL-table and AW-table lead takes a lot of memory and time because the new data often small.

4. Conclusions and future work

This article reviewed the IWEI, FUP-ERASABLE, LINE and WEIL algorithms for mining EIs for dynamic incremental databases. The theory behind each algorithm was described and weaknesses were discussed. In

Main procedure: *WEIL*
Input: a set of database, ***PDB = {PDB_0, PDB_1,…}***, the weight information for earch item, ***W***, and a maximum gain threshold, ***MGT***
Output: a set of weighted erasable itemsets, ***R***
Variables: a WEIL-table, ***WT***, a set of WEI-list, ***WEI***, a AW-table, ***AW***
WT = ∅, WEI = ∅, AW = ∅
For each PDB_i in *PDB*, **do**
  Call ***Construct_PDB(WT, WEI, AW, PDB_i)***
  If a mining request occurs, **Then**

Calculate MGT
For each itemset *wei* in WEI, **do**
If *(wei.Gain/ AW(wei)) ≤ MGT,* **Then** ***R = R ∪ wei***
End for
Call ***Mine_WEP(WT, WEI, AW, MGT, R)***
Return ***R***
R = ∅
End if
End for

a) Main procedure of WEIL algorithm

Sub procedure: ***Construct_PDB***
Input: A WEIL-table, **WT**, a set of WEI-list, **WEI**, a AW-table, **AW**, a product database, **PDB_i**
Output: WT, **WEI** and **AW** update with PDB_i
Variables: total weight, W_p, the frequency of item, ***Sup***
For each product P_k in PDB_i, **do**
For each a new element, ***e***, **do** ***WT ← e***, ***e.PIDs = P_k.PID***, ***Gain(e) = P_k.Val***
$W_p = 0$
For each item i_k in P_k, **do** $W_p = W_p + W(i_k)$
For each item i_k in P_k, **do**
If w*ei* of i_k already exists WEI, **Then**
wei.PIDs = w**ei.PIDs** ∪ P_k**.PID, Gain(wei) = Gain(wei) + P_k.Val**
Else
W*EI* ← *wei'* ,with wei' be a new EI-list for i_k
wei'.PIDs = P_k.PID, Gain(wei') = P_k.Val
End if
***Sup* = |*wei.PIDs*|, $AW(i_k) = (AW(i_k) \times (Sup-1) + W(i_k)/W_p)/Sup$**
End for
End for

Sub procedure: ***Mine_WEP(WT, WEI, AW, MGT, R)***
Input: A WEIL-table, **WT**, a set of WEI-list, **WEI**, a AW-table, **AW**
Output: ***R***
WEP = ∅, MAW = the maximum value in AW
For each item I_i in WEI or WEP
For each item I_j in WEI or WEP //i<j
wep = ∅, wep.Itemset = I_i.Item ∪ I_j.Item // I_i.Itemset ∪ I_j.Itemset
If I_i and I_j satisfy the PRI condition, **Then**
wep.dPID = I_j.PIDs // I_j.dPIDs
Gain(wep) = Gain(I_i) + Gain(I_j)
If *(Gain(wep)/MAW) ≤ MGT*, **Then**
W*EP* = *WEP* ∪ *wep*
If(Gain(wep)/ AW(wep)) ≤ MGT, **Then** ***R = R ∪ wep***
End if
Else
wep.dPID = I_i.PIDs·I_j.PIDs *or I_i.dPIDs·I_j.dPIDs*
Gain(wep) = Gain(I_i) + Gain(wep.dPIDs)
If *(Gain(wep)/MAW) ≤ MGT*, **Then**

```
                    WEP = WEP ∪ wep
                    If(Gain(wep)/ AW(wep)) ≤ MGT, Then R = R ∪ wep
                End if
            End if
        End for
        If |WEP| > 1, do Call Mine_WEP(WT, WEP, AW, MGT, R)
End for
```

b) Sub procedure of WEIL algorithm

Figure 14. *Main procedure of WEIL*

future work, some issues related to mining EIs for dynamic databases should be studied to various dynamic itemset mining fields deal with diverse stream environments such as damped window, sliding window, pre-large concept, and time-considering stream areas.

References

[1] D. W. Cheung, J. Han, V. T. Ng, and C. Y. Wong, *"Maintenance of discovered association rules in large databases: An incremental updating approach",* The Twelfth International Conference on Data Engineering, pp. 106-114, 1996

[2] Deng ZH, Fang G, Wang Z, Xu X. *Mining erasable itemsets*. In: Proceedings of the 8th IEEE International Conference on Machine Learning and Cybernetics, Baoding, Hebei, China, 2009, 67–73.

[3] Z. Deng, G. Fang, and Z. Wang, "*MINING ERASABLE ITEMSETS*", Proceedings of the 8th International Conference on Machine Learning and Cybernetics, vol. 1, pp. 67-73, Jul. 2009.

[4] Z. Deng and X. Xu, "*An Efficient Algorithm for Mining Erasable Itemsets*", Advanced Data Mining and Applications, pp. 214-225, Nov. 2010.

[5] Z. Deng and X. Xu, "*Fast mining erasable itemsets using NC_sets*", Expert Systems with Applications, vol. 39, no. 4, pp. 4453-4463, Mar. 2012.

[6] T. Le, B. Vo, and F. Coenen, "*An Efficient Algorithm for Mining Erasable Itemsets Using the Difference of NC-Sets*", IEEE International Conference on Systems, Man, and Cybernetics, pp. 2270- 2274, Oct. 2013.

[7] T. Le and B. Vo, "*MEI: An efficient algorithm for mining erasable itemsets*", Engineering Applications of Artificial Intelligence, vol. 27, pp. 155-166, Jan. 2014.

[8] 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, 2016.

[9] T. P. Hong, K. Y. Lin, C. W. Lin, and B. Vo, "*An incremental mining algorithm for erasable itemsets*", IEEE International Conference on innovations in intelligent systems and applications, pp. 286–289, 2017.

[10] G. Lee, and U. Yun, "*Single-pass based efficient erasable pattern mining using list data structure on dynamic incremental databases*", Future Generation Computer Systems, 80, 12–28, 2018.

[11] H. Nam, U. Yun, E. Yoon, J. C. W. Lin, "*Efficient approach for incremental weighted erasable pattern mining with list structure*". Expert Syst. Appl. 143,20