Mining Weighted Erasable Itemsets Over the Incremental Database Based on the InvP-List

Ye-In Chang, Siang-Jia Du, and Chin-Ting Lin

***Abstract*—An erasable itemset is the low profit itemset in the product database. The previous algorithms for mining erasable itemsets ignore the weight of each component of the product and mine erasable itemsets by concerning the product profit only in static product databases. But, when we consider the weight of each component, previous algorithms for mining weighted erasable itemsets would violate the anti-monotone property. That is, the subset X of an erasable pattern Y may not be an erasable pattern. The IWEI algorithm uses the static overestimated factor of itemsets profits to satisfy the "anti-monotone property" of weighted erasable itemset and constructs the IWEI-Tree and OP-List data structure for the dynamic database. However, the IWEI-Tree has to be reconstructed, when reading the whole product database is finished. It will take long time to complete the mining of the whole tree, if the database is frequently updated. The IWEI algorithm generates the too low static value of the overestimated factor to prune candidates. To solve those problems, in this paper, we propose the Inverted-Product-List algorithm (InvP-List) and with the local estimated factor to identify weighted erasable itemsets candidates from the Candidate-List which is generated from InvP-List. We propose the appropriate estimated factor to reduce the number of candidates which is called LMAW. LMAW is a local estimated factor which is used to check whether the itemset is a weighted erasable itemset or not. Our InvP-List algorithm also requires only one database scan. Moreover, our proposed algorithm concerning the local estimated factor creates few numbers of candidates than the IWEI algorithm. From the performance study, we show that our InvP-List algorithm is more efficient than the IWEI algorithm both in the real and the synthetic datasets.**

***Index Terms*—Erasable itemset, frequent patterns, itemset pruning, local estimated factor, weight constraint.**

I. Introduction

Erasable itemset mining is an approach for mining itemsets with low profits to be erasable from large-scale databases of products in manufacturing industries, when the manager of manufacturing industries faces financial crises [1]-[4]. In other words, an erasable itemset is a component set of products with the low profit so that we will not develop such products which contain discarded components.

In previous algorithms [1]-[3], [5], [6] they only consider the case that the gain value of itemset X is an erasable itemset, when the gain value of itemset X is not larger than the threshold value. The definition of the threshold value is a percentage value δ (i.e., the threshold which is given by the user) multiplied by the total profit value of products in the product database. The definition of the gain value of the itemset X, gain(X), is a summation of profits of all the products which contain one or more items of itemset X.

Fig. 1 shows an example of a product database TD1, where the threshold $\delta = 0.3$, the threshold value = 1050 (= (700 + 200 + 2000 + 600) × 0.3), and gain(A) = 900 (= 700 + 200). Due to that the gain value of item A is smaller than the threshold value 1050, we can conclude that item A is an erasable item in the product database TD1.

Product	Item	Profit
P_1	*A, D, E*	700
P_2	*A, D*	200
P_3	*C*	2000
P_4	*D, E*	600

Fig. 1. An example of the product database TD1.

The previously traditional methods [2]-[4], [6] of erasable itemset mining have several limitations in terms of data accumulation and features of each item. First, previous algorithms are designed for processing for the static database. Once new data is inserted into the database, they must take time to scan the database more than once. However, the amount of the data of products increases due to the activation and growth of the components in the real life. Due to such a reason, by using previous algorithms to deal with the incremental database [4], [7]-[9] is not efficient for facing such a change.

There are four cases in the incremental database environment. When the new data is inserted into the database, the total profits and the threshold value of the original data will be changed. Therefore, the erasable itemset existing in the original database may or may not be the erasable itemset. Moreover, the inerasable itemset existing in the original database may or may not be the inerasable itemset. Consequently, those situations can be classified into 4 cases by the characteristic of data accumulation. Therefore, once the database is upgraded, data must be examined again.

Moreover, previous algorithms regard that all the components which compose products as the same importance, no matter what types of components which they are going to mine. But each product is constituted of various components, those components have different features in their products. So, each component has its own weight [4], [10]-[13] such as the price in the real world.

In the real world, Lee *et al.* [4] propose an algorithm to mine weighted erasable itemsets called the IWEI algorithm which considers the weight of each item such as the price in

Manuscript received December 13, 2021; revised April 11, 2022.

The authors are with National Sun Yat-sen University, Taiwan (e-mail: changyi@mail.cse.nsysu.edu.tw, jacky83528@gmail.com, rsps1008@g-mail.nsysu.edu.tw).

the real world, instead of the frequency of the item in the database only. First, the IWEI algorithm constructs the tree structure to store the information of the data by scanning the database just one time. But the IWEI algorithm must repeatedly reconstruct the tree according to the descending order of the frequency of items for compacting the tree. Second, the IWEI algorithm uses an estimated factor called MAW to reduce the computational cost. However, the MAW value of this fixed estimated factor for pruning invalid candidates of any size is so low, which results in too many candidates. So, we propose the Inverted-Product-List (InvP-List) algorithm, to discover the weighted erasable itemsets based on the profit of each product and the weight of each item. In our data structure, we make an InvP-List to record the information of items of length 1 and use this list to generate a 1-Candidate list to mine weighted erasable itemsets. Moreover, we propose the local estimated factor to prune the number of candidates which is called LMAW. The value of LMAW would be changed according to the length of the itemset, instead of the fixed value. Then, we find the maximum average weight of items of length 1 as LMAW and use LMAW and the gain value of the item to calculate the value of MGain to check whether an item is a candidate or not. If the item is a candidate, we will construct 1-Candidate list to store it. After finding 1-Candidate list, we check whether an item is a real weighted erasable itemset in 1-Candidate list or not. Then, we find a new LMAW for itemset of length 2 and generate candidate itemsets of length 2 based on 1-Candidate list. We will find the weighted erasable itemset of the long length continuously until we cannot find any more candidates of the long length.

Our list structure can decrease the time of reconstructing the tree and our algorithm can reduce the number of invalid candidates which results in decreasing the processing time. From the simulation result, we show that our proposed InvP-List algorithm provides better performance than the IWEI algorithm.

The rest of paper is organized as follows. In Section II, we give a survey of the IWEI algorithm. In Section III, we present our proposed approach. Section IV presents the performance study of our approach and makes a comparison between our approach and the IWEI algorithm. Finally, we give a conclusion in Section V.

II. A Survey of the IWEI Algorithm

The IWEI algorithm was proposed by Lee *et al*. [4] and it constructs an original IWEI-Tree by scanning an original product database in Fig. 2, then reconstructs the tree in descending order of frequency of item for compacting the tree structure. Once the new product information is inserted into the original IWEI-Tree, then the algorithm reconstructs the tree for the purpose of a compact tree for maintaining.

Product	Items	Profit
P_1	*a*, *d*, *e*	1000
P_2	*a*, *c*, *d*, *e*, *g*	200
P_3	*f*, *h*, *i*	200
P_4	*d*, *e*	2000
P_5	*a*, *c*, *f*, *i*	300
P_6	*c*, *d*, *e*	400

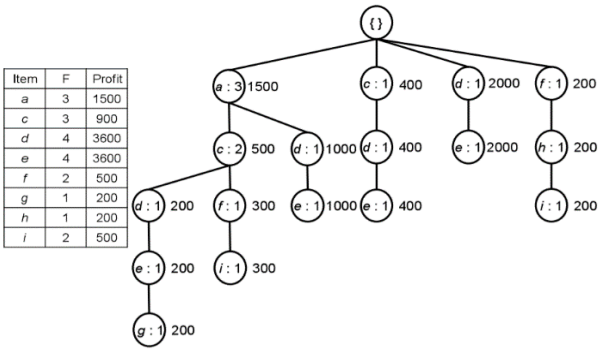
Fig. 2. An example for example product database.

By concerning the static databases, there is a problem in which they must read data straightway within a single database scanning in incremental database environments. Because the circumstances of such incremental databases are constantly altering according to the accumulation of information.

For the purpose of conducting the mining process more efficiently, they propose a new list data structure which reduces the duplicated information, called Order-and-profit-list (OP-list).

An IWEI-Tree consists of a header table and a prefix tree. The header table has three columns: item name, frequency(F), and profit. The prefix tree is composed of multiple nodes, where each node includes name of item, frequency, total profit, parent link, children set, pre-order-index, and res-flag, res-flag is used to check whether each node has been restructured or not yet.

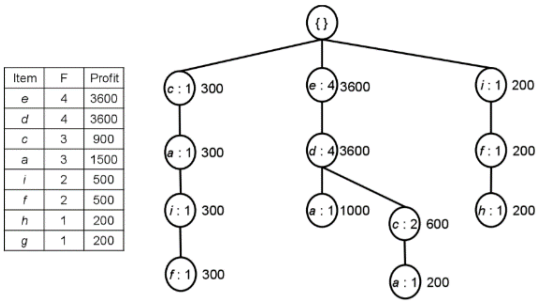
First, they construct the IWEI-Tree according to the lexicographic order of item. For example, products P1-P6 have been inserted into the IWEI-Tree as shown in Fig. 3.



Item	F	Profit
a	3	1500
c	3	900
d	4	3600
e	4	3600
f	2	500
g	1	200
h	1	200
i	2	500

Fig. 3. The IWEI-Tree after inserting P1-P6.

Second, the IWEI algorithm conducts the reconstructing of the tree according to descending order of frequency in order to efficiently store product information. Fig. 4 shows the reconstruction of the IWEI-Tree according to descending order of frequency.



Item	F	Profit
e	4	3600
d	4	3600
c	3	900
a	3	1500
i	2	500
f	2	500
h	1	200
g	1	200

Fig. 4. The reconstruction of IWEI-Tree according to descending order of frequency.

After constructing the IWEI-Tree, OP-list is generated, where the list is composed of pre-order index, total profit, and post-order index information of the nodes which composes the tree. Fig. 5 shows the result of restructuring the IWEI-Tree in Fig. 4.

Pre-order	1	2	3	4	5	6	7	8	9	10	11	12	13
Profit	300	300	300	300	3600	3600	1000	600	200	200	200	200	200
Post-order	4	3	2	1	10	9	5	8	7	6	13	12	11

Fig. 5. An OP -list of restructuring the IWEI-Tree.

However, containing information of all the nodes into OP-

list can be so inefficient, because there are not the whole nodes in the tree participating in the mining process. OP-list just keeps nodes information which is contained in erasable 1-itemsets.

They use an overestimated factor MGain(X) to check whether itemset X is an erasable itemset candidate or not. If MGain(X) > threshold, then itemset X is not a candidate, where MAW means the largest value of AW in the itemsets with length 1. MGain(X) is defined as Gain(X) divided by MAW.

After passing the check of MGain(X), then they use the AW result. They can obtain a weighted gain of X, WGain(X). If WGain(X) > threshold, itemset X is not a weighted erasable itemset, vice versa, WGain(X) is defined as Gain(X) divided by AW(X).

III. The Proposed Algorithm

A. Data Structure

To efficiently find the weighted erasable itemsets, the data structure must avoid the reconstructing process, and the value of the estimated factor is appropriate to decrease the number of candidates. In this section, we propose an Inverted-Product List (InvPL) Algorithm to efficiently discover the weighted erasable itemsets by using the inverted file information of the product database as the foundation of our data structure.

In weighted erasable itemsets mining, it considers that each item has the respective profit in the product. The weighted erasable itemsets are the compositions of items which do not contribute the most valuable profit in the product database. Moreover, the "anti-monotone property" does not hold in weighted erasable itemsets mining. Therefore, we propose the InvPL algorithm for mining the inverted product information and consider the local estimated factor to hold the "anti-monotone property". The InvP-List mining operation can be efficiently executed by using the item information of each list node sorted in the InvP-List. In this section, we first describe the Inverted-Product table that is used to compose the product database. Then, we present the InvP-List data structure which is based on the Inverted-Product table to find weighted erasable itemsets efficiently.

	Product	Item	Profit
Original	P_1	A, B, D	600
	P_2	A, D	200
	P_3	C	2100
	P_4	B, D	300
	P_5	F, G	400
	P_6	D, E, F	1500
Incremental	P_7	G, H	600
	P_8	F, H	500

Fig. 6. An example of product database TD1.

Because different products may contain common items, those products can be composed in a table which is used to transform the product-base table into the item-based table. The InvPL algorithm uses the inverted-product based algorithm to efficiently discover the weighted erasable itemsets in the product database. Fig. 6 and Fig. 7 show an example of product database TD1 and the related weight table WT1, respectively. We simply establish a table of items with the corresponding products and the related information which is denoted as InvP-Table as shown in Fig. 8.

Item	Weight
A	0.8
B	0.8
C	0.9
D	0.3
E	0.9
F	0.5
G	0.8
H	0.7

Fig. 7. Weight table WT1.

Item	Pid	Gain	AW	Count
A	[1, 2]	800	$[\frac{0.8}{1.9}, \frac{0.8}{1.1}]$	2
B	[1, 4]	900	$[\frac{0.8}{1.9}, \frac{0.8}{1.1}]$	2
D	[1, 2, 4, 6]	2600	$[\frac{0.3}{1.9}, \frac{0.3}{1.1}, \frac{0.3}{1.1}, \frac{0.3}{1.7}]$	4
C	[3]	2100	$[\frac{0.9}{0.9}]$	1
F	[5, 6]	1900	$[\frac{0.5}{1.3}, \frac{0.5}{1.7}]$	2
G	[5]	400	$[\frac{0.8}{1.3}]$	1
E	[6]	1500	$[\frac{0.9}{1.7}]$	1

Fig. 8. The recorded InvP-Table of product database TD1.

For each item I, the InvP-Table contains product identification (Pid), gain value (Gain), average weight (AW) of item I in the related product and count. The following InvP-List is constructed according to the InvP-Table which requires only the scan of product database one time. The InvP-Table records the following information of each item I:

- Pid: It represents the set of products which contains item I in the product database.
- Gain: It is constituted of the product profit. It represents the sum of the profit of item I in the related products of the database.
- AW: It represents the set of the average weight which contains item I in the related products.
- Count: It represents the appearance of item I in the product database.

Let ISet = {*I*1, *I*2, ..., *In*} be a set of all items in the product database, DB = {*P*1, *P*2, ..., *Pm*} be a product database which is composed of products and *X* = {*I*1, *I*2, ..., *Is*} be an itemset, s ≥ 1.

The gain value of itemset (Gain) shows the amount of the profits related to each product in the database. Given an itemset X, gain of X, Gain(X), is a summation of profits of all the products that contain one or more items of X, which is computed as follows [4]:

$$\text{Gain}(X) = \sum_{r=1}^{m} \{Inter(X, P_r) * P_r.\text{profit}\},$$

$$\text{Inter}(X, P_r) = \begin{cases} 1, & \text{if } X \cap P_r \neq \emptyset \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

The value of Gain(X) could only be increased, instead of being decreased, as the data is increased.

The AW(Ip) shows the amount of the average weight

values related to each item in products. AW(Ip) is used to compute the ratio of the weight of item I to the sum of all the weight values for each product P_j, which is defined as follows [4]:

$$\mathrm{AW}(I_p) = \frac{\sum_{j=1}^{m}\{\frac{w(I_p)}{\sum Ip \in P_j \cdot w(I_r)}\}}{k} \quad (2)$$

Then, an average weight of itemset X, AW(X), is computed as follows:

$$\mathrm{AW}(X) = \frac{\sum_{p=0}^{s} Aw(i_p)}{s} \quad (3)$$

The LMAW is the local maximum average weight of the item of length k, which is used to compute the estimated value to prune candidates, the value of LMAW is between 0 and 1. By sorting the count in the descending order, we can obtain LMAW and check if any item does not only appear one time but also only appear in a product. Note that for such a special case, we will introduce the special case later.

For example, item A appears in products P1 and P2, totally 2 products. Gain(A) is 800 (= 600 + 200), and AW(A) is $\frac{0.8}{1.9}, \frac{0.8}{1.1} (= \frac{0.8}{0.8+0.8+0.3}, \frac{0.8}{0.8+0.3})$.

The MGT is multiplied by the user-defined threshold and the summation of the profit in the product database. If MGain(X) is larger than the MGT, itemset X will not be a candidate.

The maximum gain of itemset X, MGain(X), is computed as follows:

$$\mathrm{MGain}(X) = \frac{\mathrm{Gain}(X)}{LMAW} \quad (4)$$

That is, for those candidates X which have the value of MGain(X) larger than MGT, they are what we can prune.

After sorted by the count of length 1 items in the initial InvP-Table as shown in Fig. 9, the InvP-Table will be recorded according to the descending order of the count. But if items have the same count, it will be recorded according to the alphabetical order.

Item	Pid	Gain	AW	Count
D	[1, 2, 4, 6]	2600	$[\frac{0.3}{1.9}, \frac{0.3}{1.1}, \frac{0.3}{1.1}, \frac{0.3}{1.7}]$	4
A	[1, 2]	800	$[\frac{0.8}{1.9}, \frac{0.8}{1.1}]$	2
B	[1, 4]	900	$[\frac{0.8}{1.9}, \frac{0.8}{1.1}]$	2
F	[5, 6]	1900	$[\frac{0.5}{1.3}, \frac{0.5}{1.7}]$	2
C	[3]	2100	$[\frac{0.9}{0.9}]$	1
E	[6]	1500	$[\frac{0.9}{1.7}]$	1
G	[5]	400	$[\frac{0.8}{1.3}]$	1

Fig. 9. The sorted InvP-Table of product database TD1.

On the other hand, if there exists the special case, e.g., LMAW = 1, then we can check whether such an item I will be the candidate or not by using MGain(X) in this case. Otherwise, we will not accept it and continuously search the new LMAW.

In Fig. 9, item C is a special case in the product database. Item C only appears in product P3 and product P3 contains only one item (i.e., C), so the average weight of item C is 1 and LMAW is 1 (= AW(C)). Let the threshold be 32%, the MGT of the original product database is 1632 (= 5100 × 0.32). We have Gain(C) = 2100 and LMAW = 1, so the MGain(C) is 2100. Because the MGain(C) is 2100 which is larger than MGT, item C will not be a candidate of length 1 item.

Due to item C will not be the candidate of length 1 item, we must continuously search the new LMAW. Finally, we find the new LMAW = 0.61 (= AW(G)) in InvP-Table.

After sorting the InvP-List, we can determine LMAW. Then, we create the candidate of length 1 by MGain(X). By using 1-Candidate table to mine weighted erasable itemsets, we can reduce the search time, and can solve the accumulated problem. If we prune the items in InvP-List, the weighted erasable of length k may lose, when new products are inserted into the product database. However, we prune items in 1-Candidate table which does not cause missing cases, when we process product database update. Because 1-Candidate table is recorded again while we process the update of the product of database. 1-Candidate table contains four columns, item, Pid, Gain and average weight (AW). We construct 1-Candidate table by InvP-List related to product database TD1 as shown in Fig. 10. We use Gain of each item in InvP-List and LMAW to calculate MGain(X). If MGain(X) is not larger than MGT, we record the item name, Pid, Gain and AW in 1-Candidate table.

Item	Pid	Gain	AW
[A]	[1, 2]	800	0.57
[B]	[1, 4]	900	0.57
[G]	[5]	400	0.61

Fig. 10. 1-Candidate table for TD1.

In Fig. 10, we have MGT = 1632 and LMAW = 0.61 in the original product database TD1 and Gain of item A is 800. The MGain(A) = 1311 is smaller than MGT, so item A is a candidate of length 1. We record the item name, Pid, Gain and the average weight of item A in InvP-List to 1-Candidate table.

B. The Mining Algorithm with 1-Candidate-Table

After we finish the construction of the 1-Candidate table which is based on the InvP-List, we start the mining step. We perform the weighted erasable itemset mining by 1-Candidate table, instead of InvP-List. Because mining weighted erasable itemset with 1-Candidate table can reduce the search time and solve the accumulate problem. First, we check the candidates of length k. If it passes the condition, it must be a member of the weighted erasable itemset of length k. Second, we generate the candidates of length (k+1) to find the weighted erasable itemset of length (k+1) until we cannot generate any longer length candidates.

We use the weighted gain WGain(X) to check whether the candidate itemset is a real weighted erasable itemset or not. Therefore, we can find how much effectiveness of itemset X on the products which contains itemset X by WGain(X). The weighted gain of itemset X, WGain(X), is computed as follows:

$$\mathrm{WGain}(X) = \frac{\mathrm{Gain}(X)}{AW(X)} \quad (5)$$

By dividing Gain(X) with AW(X), we get the proportion

of Gain(X) to AW(X). If the MGain(X) is not larger than MGT, itemset X is a weighted erasable itemset. That is, itemset X could not make many profits, and vice versa.

The value of AW(X) may be increased or decreased, as the data is increased. Similarly, the value of WGain(X) may be increased or decreased, as the data is increased.

In fact, the value of LMAW is kept as lower as better in terms of concerning the number of candidates. The reason is that when the value of LMAW is small, it represents that the denominator of formula MGain(X) = Gain(X) / LMAW is large. It will result in a large quotient. Moreover, it will result in a large MGain(X). Note that the value of MGain(X) is always smaller than or equal to WGain(X), i.e., the real value of itemset X. Therefore, if the value of MGain(X) is as near as the value of WGain(X), then we can have a large chance to prune it, which can result in a small number of candidates.

Item	Pid	Gain	AW
[A]	[1, 2]	800	0.57
[B]	[1, 4]	900	0.57
[G]	[5]	400	0.61

Fig. 11. 1-Candidate Table for checking item A.

For example, in Fig. 11, item A is a candidate of length 1. We calculate WGain(A) by Gain(A) and AW(A). WGain(A) = 1403 (= 800 / 0.57) is smaller than MGT (= 1632), so item A is a real weighted erasable itemset of length 1.

After checking the candidates of length 1, we find items A, B and G are real weighted erasable itemsets of length 1. Because their weighted gain values are smaller than MGT.

Then, we use candidates of length 1 to generate candidates of length 2. First, we select the candidate that the size of Pid is largest to combine with other candidates of length 1. Second, we use MGain(X) to reduce the number of candidates. Finally, we use WGain(X) to check the candidate which passes the WGain(X) no matter whether it is real weighted erasable itemset or not.

In order to generate candidates of length k, where k is larger than 1, we make use of two properties. First, we use the difference Pid (DiffS) property [3]. Given itemset X and itemset Y, DiffS(X, Y) is defined as follows:

$$\text{Diffs}(X, Y) = Pid(Y) \setminus Pid(X) \quad (6)$$

For example, the pid set of itemset X = [2, 3, 6, 8] and the pid set of itemset Y = [1, 2, 6], DiffS(X, Y) = [1].

Second, by using the difference Pid (DiffS) property, we use a property which is proposed from the MEI algorithm [3], and is defined as follows:

$$\text{Gain}(XY) = Gain(X) + \sum_{k \in Diffs(X,\ Y)} k.profit \quad (7)$$

For example, Fig. 12 shows DiffS(A, B) = Pid(B) \ Pid(A), and DiffS(A, B)= [4]. We can calculate the gain value of itemset [A, B] by the summation of the gain value of item A and the profit of difference Pid, Gain(AB) = 800 + 300 = 1100.

Item	Pid	Gain	AW
[A]	[1, 2]	800	0.57
[B]	[1, 4]	900	0.57
[G]	[5]	400	0.61

Fig. 12. Gain value of itemset AB.

Item	dPid	Gain	MGain
[A, B]	[4]	1100	1864
[A, G]	[5]	1200	2033

Fig. 13. Item A combining with items B and G to generate candidates of length 2.

For example, in Fig. 11, after checking the case of the real weighted erasable of length 1 from candidates, we select item A to combine with other candidates which pass the MGain(X). Item A combines with items B or G to generate candidates of length 2 which are itemsets [A, B], and [A, G] in Fig. 13. As compared to 1-Candidate table shown in Fig. 10, we record dPid to calculate Gain of itemset, instead of Pid. Because we use the concept of DiffS. Due to this reason, we will not record duplicated information of Pid.

At this time, we would calculate LMAW for itemsets of length k when k > 1. We select the first largest AW and the second largest AW of item until the k-th largest of length 1 to get AW as the new LMAW.

First, the new LMAW is 0.59 (= (0.57 + 0.61) / 2), we can find the maximum weighted gain value of itemsets [A, B] = 1864 and [A, G] = 2033. Both itemsets [A, B] and [A, G] with the maximum weighted gain value which are larger than MGT (= 1632). Therefore, itemsets [A, B], and [A, G] are not candidates of length 2, and we prune them from candidates of length 2. Because itemsets [A, B], and [A, G] will never be the real weighted itemsets no matter what items combine with them in the original product database.

Due to that [A, B], and [A, G] are not candidates of length 2, we cannot find the candidates which length is larger than 2 when we consider the case of length k that item A combine with the other items. Then, we consider a remaining candidate of length 1 to combine with the other candidates of length 1, except item A which size of Pid of candidates is the second largest. In addition, we must check LMAW of the remaining candidates. Because some candidates have been considered, like item A, those items will not be combined with the other candidates.

For example, item A has been considered already, we select another candidate of length 1, item B that the size of Pid of remaining candidates of length 1 is the second largest. We find LMAW = 0.59 (= (0.57 + 0.61) / 2). The gain value of itemset [B, G] = 1300 and LMAW = 0.59. MGain(B, G) = 2203 is larger than MGT (= 1632). Itemset [B, G] is not a candidate of length 2, so that we cannot generate candidates which length is larger than 2 when considering item B in Fig. 14.

Item	dPid	Gain	MGain
[B, G]	[5]	1300	2203

Fig. 14. Item B combining with item G to generate candidates of length 2.

Finally, we can find that items A, B and G are real weighted erasable itemsets from candidates. On the other hand, we only generate 6 candidates, but the IWEI algorithm generates 11 candidates for the same product databases.

1) Insertion of the new data

When the new product information is inserted into products, we have to check the following cases. If this product contains new item which is not recorded in the

original inverted-product table, then we record the information of Item, Pid, Gain, AW and Count of the item into the last part of the inverted-product table. Otherwise, we update the item information which is corresponding to the item in the original inverted-product table. When we finish inserting the new product information into the inverted-product table, we sort the additional product database and find an appropriate LMAW at the same time.

Due to that the new information of items are added to the last part of the inverted-product table, we cannot reconstruct the inverted-product table at all. However, we must find an appropriate LMAW. Because the new data is generated, and the AW information of the item could be changed and infected.

For example, Fig. 15 shows that product P7 is inserted into the bottom of inverted-product table and updating the corresponding items in the original inverted-product table, when product P7 comes. Due to that product P7 contains items G and H, and item G has already in the original inverted-product table, so we just add new pid to Pid set. Then, we add the new average weight to AW set and accumulate Gain, Count for the corresponding item G in the original inverted-product table.

Item	Pid	Gain	AW	Count
D	[1, 2, 4, 6]	2600	$[\frac{0.3}{1.9}, \frac{0.3}{1.1}, \frac{0.3}{1.1}, \frac{0.3}{1.7}]$	4
A	[1, 2]	800	$[\frac{0.8}{1.9}, \frac{0.8}{1.1}]$	2
B	[1, 4]	900	$[\frac{0.8}{1.9}, \frac{0.8}{1.1}]$	2
F	[5, 6]	1900	$[\frac{0.5}{1.3}, \frac{0.5}{1.7}]$	2
C	[3]	2100	$[\frac{0.9}{0.9}]$	1
E	[6]	1500	$[\frac{0.9}{1.7}]$	1
G	[5, 7]	1000	$[\frac{0.8}{1.3}, \frac{0.8}{1.5}]$	2
H	[7]	600	$[\frac{0.7}{1.5}]$	1

Product	Item	Profit
P_1	A, B, D	600
P_2	A, D	200
P_3	C	2100
P_4	B, D	300
P_5	F, G	400
P_6	D, E, F	1500
P_7	G, H	600
P_8	F, H	500

Fig. 15. Insertion of product P7 into InvP-Table.

Fig. 16 shows the final result, when products P7 and P8 are inserted into the original product database. After updating the inverted-product table, we should find the new LMAW which plays an important role to efficiently prune candidates and sort the new inverted-product table according to Count of the item at the same time.

Item	Pid	Gain	AW	Count
D	[1, 2, 4, 6]	2600	$[\frac{0.3}{1.9}, \frac{0.3}{1.1}, \frac{0.3}{1.1}, \frac{0.3}{1.7}]$	4
A	[1, 2]	800	$[\frac{0.8}{1.9}, \frac{0.8}{1.1}]$	2
B	[1, 4]	900	$[\frac{0.8}{1.9}, \frac{0.8}{1.1}]$	2
F	[5, 6, 8]	2400	$[\frac{0.5}{1.3}, \frac{0.5}{1.7}, \frac{0.5}{1.2}]$	3
C	[3]	2100	$[\frac{0.9}{0.9}]$	1
E	[6]	1500	$[\frac{0.9}{1.7}]$	1
G	[5, 7]	1000	$[\frac{0.8}{1.3}, \frac{0.8}{1.5}]$	2
H	[7, 8]	1100	$[\frac{0.7}{1.5}, \frac{0.7}{1.2}]$	2

Product	Item	Profit
P_1	A, B, D	600
P_2	A, D	200
P_3	C	2100
P_4	B, D	300
P_5	F, G	400
P_6	D, E, F	1500
P_7	G, H	600
P_8	F, H	500

Fig. 16. The insertion of Product P_8 into *InvP*-Table.

After updating the inverted-product table, we should find the new LMAW which plays an important role to efficiently prune candidates and sort the new inverted-product table according to Count of the item at the same time.

Fig. 17 shows the sorted inverted-product table which has inserted the additional information of the item into the inverted-product table. We can get the new LMAW after sorting. We have LMAW = 0.57 (= AW(G)). The value of LMAW changes, because item G is a component in product P7. Moreover, the new average weight of G which gets the value of adding the original average weight of G and the average weight of G in product P7. The average weight of length 1 item may change, when the new product is inserted. Moreover, this value is possible to be increased or decreased. Due to this reason, we must find an appropriate LMAW, because LMAW is the local maximum average weight the length 1 item average weight.

Item	Pid	Gain	AW	Count
D	[1, 2, 4, 6]	2600	$[\frac{0.3}{1.9}, \frac{0.3}{1.1}, \frac{0.3}{1.1}, \frac{0.3}{1.7}]$	4
F	[5, 6, 8]	2400	$[\frac{0.5}{1.3}, \frac{0.5}{1.7}, \frac{0.5}{1.2}]$	3
G	[5, 7]	1000	$[\frac{0.8}{1.3}, \frac{0.8}{1.5}]$	2
H	[7, 8]	1100	$[\frac{0.7}{1.5}, \frac{0.7}{1.2}]$	2
A	[1, 2]	800	$[\frac{0.8}{1.9}, \frac{0.8}{1.1}]$	2
B	[1, 4]	900	$[\frac{0.8}{1.9}, \frac{0.8}{1.1}]$	2
C	[3]	2100	$[\frac{0.9}{0.9}]$	1
E	[6]	1500	$[\frac{0.9}{1.7}]$	1

Fig. 17. InvP-Table for original + incremental product database.

When getting the new LMAW and finishing sorting the new inverted-product table, we can get the length 1 candidate table for the new inverted-product table (i.e., the original + the incremental product databases). After getting candidate table of length 1, we can use the information of items of length 1 in this table to calculate candidates and real weighted erasable itemsets by LMAW, MGain(X) and WGain(X).

C. Comparison

After all, we make a comparison to discuss the difference between our proposed InvPL algorithm and the IWEI algorithm. We use product database TD1 and weight table WT1 as an input as shown in Fig. 6 and Fig. 7 to compare the generated number of candidates and data structure of the InvPL algorithm and the IWEI algorithm.

When products P1 - P6 are scanned, the IWEI-Tree of the IWEI algorithm must reconstructed one times to compress the data structure, but our InvPL algorithm does not reconstruct data structure. Another aspect, we compare the number of candidates of the IWEI algorithm and the InvPL algorithm in the original section of product database. Because we propose the local estimated factor to rise the value of MGain so that we can prune many more unnecessary candidates. The number of candidates of the InvPL algorithm is 6, which is smaller than 11 of the IWEI algorithm.

We make a comparison of construct times and the number of candidates between our InvPL algorithm and the IWEI algorithm, when incremental data is inserted into the original product database. Our InvPL algorithm does not reconstruct the data rather than reconstruct one time in the IWEI algorithm.

Finally, we make a comparison of the number of generated candidates and the number of real weighted erasable itemsets between the InvPL algorithm and IWEI algorithm. We find the number of candidates of our InvPL algorithm (=10) is less than that of the IWEI algorithm (=18), and the number of real weighted erasable itemsets is the same (=4).

IV. Performance

In our performance study, we consider the real dataset

(http://fimi.ua.ac.be/data/) and the synthetic dataset. In this section, we will compare the performance between the IWEI algorithm and our InvPL algorithm. We will compare the processing time of those two algorithms mining for the real dataset and the synthetic datasets. We also compare the number of candidates of those two algorithms. Therefore, we evaluate the performance of the IWEI algorithm and our InvPL algorithm in terms of the processing time under the change of the threshold. Moreover, we also evaluate the process of mining weighted erasable itemsets in terms of the processing time under the change of the number of items and the number of products in the product database. The Mushroom dataset is the information of various species of the mushroom, and it is a dense dataset. The profit of products and mushrooms are generated between 10 and 100, and the weight of items and elements are generated between 0.1 and 0.9.

The parameters used in the generation of synthetic datasets are shown in Table I. The details of the Mushroom dataset are shown in Table II. (To make a comparison between the IWEI algorithm and our InvPL algorithm, for the real dataset, we use different range of user defined percentage, i.e., threshold.) The details of synthetic datasets are shown in Table III."

TABLE I: VARIABLES

Parameter	Description
Threshold	The user defined percentage
I	The number of the items in the dataset
D	The number of the products in the dataset

TABLE II: THE DETAILS OF REAL DATASET

Dataset	T_{avg}	|D|	|I|	User Defined Percentage
Mushroom	23	8124	120	0.5 – 0.9

TABLE III: THE DETAILS OF SYNTHETIC DATASETS

Case	Dataset	T_{avg}	|D|	|I|	User Defined Percentage
1	T10.I1K. D100K	10	100,000	1,000	0.0166 – 0.017
2	T10.I?K. D100K	10	1,000 – 1,080	1,000	0.0166

Then, in Fig. 18, we show the comparison of the number of candidates for the real dataset on Mushroom under the change of the threshold. In this experiment, we set the threshold to be 0.5, 0.6, 0.7, 0.8 and 0.9. From Fig. 18, we show that the number of generated candidates of our InvPL algorithm is fewer than that of the IWEI algorithm. The reason is that the IWEI algorithm uses the too low estimated factor to prune candidates. Moreover, we find that the number of candidates of both algorithms increase, when the threshold increases. Because when the threshold increases, it will cause many candidates to pass the MGT test. However, to prune candidates, the value of the estimated factor used in our InvPL algorithm is never lager than that used in the IWEI algorithm. Therefore, our InvPL algorithm will prune more number of candidates than the IWEI algorithm. Furthermore, the Mushroom dataset, a dense dataset, the number of candidates determined by our InvPL algorithm increases slowly as the value of the threshold increases. The reason is the same as stated before.

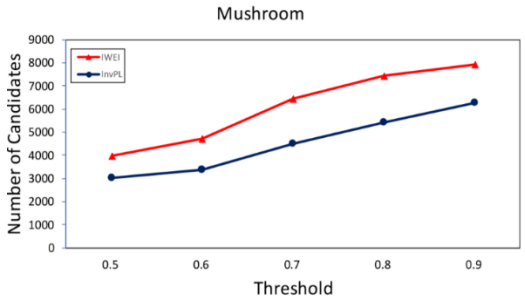


Fig. 18. A comparison of the number of candidates of the Mushroom dataset under the change of the threshold.

In Fig. 19, we show the comparison of the processing time for the real dataset, Mushroom, under the change of the threshold. In this experiment, we set the threshold to be 0.5, 0.6, 0.7, 0.8 and 0.9. From Fig. 19, we show that the processing time of our InvPL algorithm is faster than the IWEI algorithm. The reason is that the IWEI algorithm generates more number of candidates than our InvPL algorithm. Moreover, our InvPL algorithm does not reconstruct the list structure again. Therefore, our InvPL algorithm takes shorter time than the IWEI algorithm to find weighted erasable itemsets. Furthermore, we use a data structure similar to the in- verted index. Therefore, the search time for the interesting pattern is efficient and almost the fixed time, no matter what value of the threshold is and no matter what distribution of the data is.

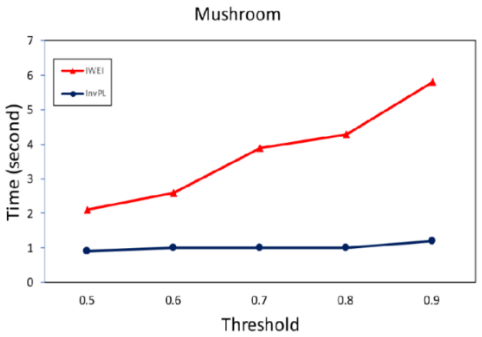


Fig. 19. A comparison of the processing time for the Mushroom dataset under the change of the threshold.

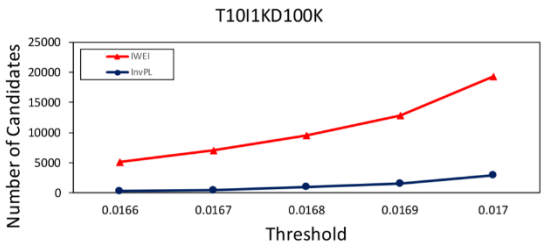


Fig. 20. A comparison of the number of candidates for the synthetic dataset, T10.I1K.D100K, under the change of the threshold

Next, we show the performance experiment of the both algorithms for the synthetic dataset. In Fig. 22, we show the number of candidates of both algorithms for the synthetic dataset, T10.I1K.D100K under the change of the threshold. In this experiment, we set the threshold to be 0.0166, 0.0167, 0.0168, 0.0169 and 0.0170. From Fig. 20, we show that the number of generated candidates of our InvPL algorithm is fewer than that of the IWEI algorithm. The reason is that the IWEI algorithm always uses the too low estimated factor to

prune candidates. Moreover, we find that the number of candidates of the IWEI algorithm increases faster than that of our InvPL algorithm, when the threshold increases. The reason is the same as what we have stated above. Therefore, the InvPL algorithm prunes more number of candidates than the IWEI algorithm.

In Fig. 21, we show the comparison of the processing time for the synthetic dataset, T10.I1K.D100K under the change of the threshold. In this experiment, we set the user defined percentage to be 0.0166, 0.0167, 0.0168, 0.0169 and 0.0170. From Fig. 21, we show that the processing time of our InvPL algorithm is shorter than the IWEI algorithm. The reason is that our InvPL algorithm generates fewer number of candidates than the IWEI algorithm. Moreover, our InvPL algorithm does not reconstruct the list structure again. Therefore, our InvPL algorithm takes shorter time than the IWEI algorithm to find the weighted erasable itemsets.

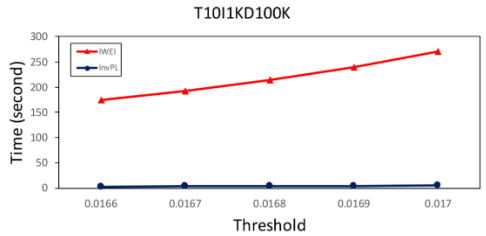


Fig. 21. A comparison of the processing time for the synthetic dataset, T10.I1K.D100K, under the change of the threshold.

In Fig. 22, we show the comparison of the processing time for the synthetic dataset, T10.I(1-1.08)K.D100K, under the change of the number of items. In this experiment, we set the user defined percentage to be 0.0166.

From Fig. 22, we show that the processing time of our InvPL algorithm is less than the IWEI algorithm. The reason is that our InvPL algorithm examines candidates more easily than the IWEI algorithm when candidates of long length. Moreover, we observe that the processing time of both algorithms increases. But the process time of our InvPL algorithm increases slower than that of the IWEI algorithm, when the number of items increases. Because our InvPL algorithm obtains candidate itemsets easily, when the number of items increases. Moreover, our InvPL algorithm does not reconstruct the list structure again. Therefore, our InvPL algorithm takes less time than the IWEI algorithm to find weighted erasable itemsets.

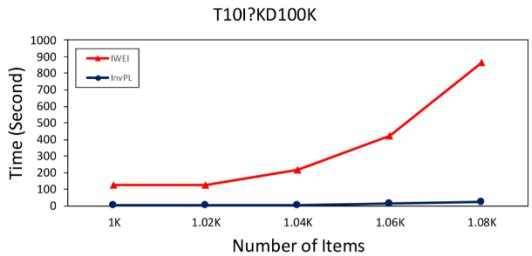


Fig. 22. A comparison of the processing time for the synthetic dataset, T10.I(1- 1.08)K.D100K, under the change of the number of items.

Therefore, our InvPL algorithm takes less time than IWEI algorithm to find weighted erasable itemsets. In our simulation study, we let the weight of each item be between 0.1 and 0.9, instead of between 0.5 and 0.75 in the performance study of the IWEI algorithm. Because we consider that it simulates to the real life. Under such a consideration, as |D| increases, the threshold value (i.e., total product profits * threshold) will also increase, which results in the case that the Gain value of each itemset also could increase. Therefore, the chance to be the weighted erasable itemset decreases, resulting in the decrease of the processing time.

V. Conclusions

In this paper, we have proposed an InvP-List algorithm which can efficiently mine weighted erasable itemsets with fewer number of candidates. We use the list structure of the inverted product dataset to record the database information of items in each product in our algorithm. Furthermore, our proposed algorithm dynamically calculates LMAW to compute the value of MGain which is an overestimated factor to prune candidates. Then, we have studied the performance of the InvP-List algorithm and the IWEI algorithm. We have conducted several experiments using the real dataset and different synthetic data. The performance results have shown that the InvP-List algorithm has better performance than the IWEI algorithm.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

Chang and Du concuted the research; Du and Lin analyzed the data and wrote the paper; all authors had approved the final version.

Acknowledgment

This research was supported in part by the Ministry of Science and Technology of Republic of China under Grant No. MOST108-2221-E110-060.

References

[1] Z.-H. Deng, G.-D. Fang, Z.-H. Wang, and X.-R. Xu, "Mining erasable itemsets," in *Proc. the Int. Conf. on Machine Learning and Cybernetics*, 2009, pp. 67–73.

[2] Z.-H. Deng and X.-R. Xu, "Fast mining erasable itemsets using NC-sets," *Expert Systems with Applications*, vol. 39, no. 4, pp. 4453–4463, 2012.

[3] T. Le and B. Vo, "MEI: An Efficient Algorithm for Mining Erasable Itemsets," *Engineering Applications of Artificial Intelligence*, vol. 27, no. 1, pp. 155–166, 2014.

[4] 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, no. 1, pp. 213–234, 2016.

[5] Z. Deng and X. Xu, "An efficient algorithm for mining erasable itemsets," in *Proc. the Int. Conf. on Advanced Data Mining and Applications*, 2010, pp. 214–225.

[6] T. Le, B. Vo, and F. Coenen, "An efficient algorithm for mining erasable itemsets using the difference of NC-sets," in *Proc. the IEEE Int. Conf. on Systems, Man, and Cybernetics*, 2013, pp. 2270–2274.

[7] U. Yun and G. Lee, "Incremental mining of weighted maximal frequent itemsets from dynamic databases," *Expert Systems with Applications*, vol. 54, no. 1, pp. 304–327, 2016.

[8] U. Yun and H. Ryang, "Incremental high utility pattern mining with static and dynamic databases," *Applied Intelligence*, vol. 42, no. 2, pp. 323–352, 2015.

[9] U. Yun, H. Ryang, G. Lee, and H. Fujita, "An efficient algorithm for mining high utility patterns from incremental databases with one

database scan," *Knowledge Based Systems*, vol. 124, no. 1, pp. 188–206, 2017.
[10] J. C.-W. Lin, W. Gan, P. Fournier-Viger, and T.-P. Hong, "RWFIM: Recent weighted frequent itemsets mining," *Engineering Applications of Artificial Intelligenc e*, vol. 45, no. 1, pp. 18–32, 2015.
[11] L. H. Son, P. L. Lanzi, B. C. Cuong, and H. A. Hung, "Data mining in GIS: A novel context-based fuzzy geographically weighted clustering algorithm," *International Journal of Machine Learning and Computing*, vol. 2, no. 3, pp. 235–238, 2012.
[12] S. Mutalib, A. Mohamed, S. Abdul-Rahman, and N. Mustafa, "Weighted frequent itemset of SNPs in genome wide studies," *International Journal of Machine Learning and Computing*, vol. 8, no. 4, pp. 311–318, 2018.
[13] L. Nguyen, G. Nguyen, and B. Le, "Fast algorithms for mining maximal erasable patterns," *Expert Systems with Applications*, vol. 124, pp. 50–66, 2019.

Ye-In Chang received her B.S. degree in computer science and information engineering from National Taiwan University, Taipei, Taiwan, in 1986. She received her M.S. and Ph.D. degrees in computer and information science from The Ohio State University, Columbus, Ohio, in 1987 and 1991, respectively. From August 1991 to July 1999, she joined the Faculty of Department of Applied Mathematics at National Sun Yat-Sen University, Kaohsiung, Taiwan. From August 1997, she has been a professor in the Department of Applied Mathematics at National Sun Yat-Sen University, Kaohsiung, Taiwan. Since August 1999, she has been a professor in the Department of Computer Science and Engineering at National Sun Yat-Sen University, Kaohsiung, Taiwan. Her research interests include database systems, distributed systems, multimedia information systems, mobile information systems and data mining.

S. J. Du received M.S. degrees in computer science and engineering from National Sun Yat-Sen University in 2018. His research interests include distribute computing and data mining. For the field of data mining, he focuses on mining the erasable pattern or the erasable itemset. He is currently a system designer in Taiwan.

C. T. Lin received the B.S. degree from National Taichung University of Education in 2018. He is currently a M.S. student in the Department of Computer Science and Engineering at National Sun Yat-Sen University in Taiwan. His research interests include data mining and distributed computing.