

# An Efficient Algorithm for Mining Erasable Itemsets Using the Difference of NC-Sets

Tuong Le

University of Food Industry  
Ho Chi Minh City, Viet Nam  
tuonglecong@gmail.com

Bay Vo

Ton Duc Thang University  
Ho Chi Minh City, Viet Nam  
bayvodinh@gmail.com

Frans Coenen

Department of Computer Science,  
University of Liverpool, UK  
coenen@liverpool.ac.uk

**Abstract**—This paper proposes an improved version of the MERIT algorithm, dMERIT+, for mining all “erasable itemsets”. We first establish an algorithm MERIT+, a revised version of MERIT, which is then used as the foundation for dMERIT+. The proposed algorithm uses: a weight index, a hash table and the “difference” of Node Code Sets (dNC-Sets) to improve the mining time. A theorem is derived first to show that dNC-Sets can be used for mining erasable itemsets. The experimental results show that dMERIT+ is more effective than MERIT+ in terms of the runtime.

**Keywords**- data mining; difference set; erasable itemsets.

## I. INTRODUCTION

Frequent itemset mining is a well-established element of data mining. Significant algorithms include: Apriori [1], Eclat [18], FP-Growth [5], FP-Growth\* [7], BitTableFI [6], Index-BitTableFI [15] and DBV-FI [16]. A variation of frequent itemset mining is Frequent Closed Itemset (FCI) mining [13, 14], example of FCI algorithms include: CLOSET [12], CLOSET+ [19], CHARM and dCHARM [18] and DBV-Miner [17].

In 2009, Deng et al. [4] first presented the problem of mining “erasable itemsets”, an interesting variation of frequent itemset mining. The example application is a factory which produces many products created from a number of items (components), each product has some income (gain) associated with it. To produce all products requires a financial resource to buy and store all required items. However, in a financial crisis situation this factory has not enough money to purchase all necessary components as usual. In this context, this task is to find the itemsets (the sets of items) which can best be eliminated (erased) so as to minimize the loss to the factory’s gain. Managers can then utilize the knowledge of these erasable itemsets to make a new production plan. Simultaneously, in [4], Deng et al. also proposed the META (Mining Erasable iTemsets with the Anti-monotone property) algorithm, based on Apriori, to solve the problem of identifying all erasable itemsets. However, the runtime of this algorithm was slow because it scanned the dataset many times and used a naïve strategy for mining erasable itemsets.

Consequently, in [3], Deng et al. proposed the VME (Vertical-format-based algorithm for Mining Erasable itemsets) algorithm which was faster than META [4]. However, VME still featured some significant disadvantages: (i) VME scans the input dataset twice (it is well established that

the scanning of dataset requires considerable computer time and memory usage, ideally a single scan is desirable); (ii) VME uses a breadth-first-search strategy in which all erasable ( $k-1$ )-itemsets will be used to create erasable  $k$ -itemsets however, classifying erasable ( $k-1$ )-itemsets with the same prefix means that the generation of erasable ( $k-2$ )-itemsets is computationally intensive; and (iii) VME stores each product’s gain in the form of a tuple,  $\langle PID, Val \rangle$ , this leads to the duplication of data because a  $\langle PID, Val \rangle$  pair can appear in many PID\_Lists of different erasable itemsets. Thus the VME algorithm requires a lot of memory usage (PID = Product ID).

MERIT (Fast Mining ERasable ITemsets) [2], is an alternative algorithm for mining erasable itemsets which uses “NC-Set structure” to reduce the memory usage (see Section 2). However this algorithm still displayed some weaknesses (see Section 3). In this paper, we propose an improved version of MERIT, dMERIT+, which uses a weight index, a hash table of erasable 1-itemsets and the difference of NC-Sets to reduce the overall runtime and the memory usage.

The rest of the paper is organized as follows. Section 2 presents the basic concepts. Section 3 shows the discussions on MERIT algorithm. dMERIT+ algorithm is proposed in Section 4 and an example of the process of dMERIT+ is presented in Section 5. Section 6 shows the results of experiments. Finally, the paper is concluded in Section 7 with a summary and some future research issues.

## II. BASIC CONCEPTS

Let  $I = \{i_1, i_2, \dots, i_m\}$  be a set of all items. Let  $DB = \{P_1, P_2, \dots, P_n\}$  be a product dataset where each product  $P_i$  ( $1 \leq i \leq n$ ) is represented by an  $\langle Y, Val \rangle$  pair, where  $Y$  is a subset of  $I$  and  $Val$  is the gain associated with  $P_i$ . An example dataset is given in Table 1 (this dataset will be used throughout the rest of this paper).

TABLE I. AN EXAMPLE DATASET

Product	Items	Val (\$)
$P_1$	$a, b$	1,000
$P_2$	$a, b, c$	200
$P_3$	$b, c, e$	150
$P_4$	$b, d, e, f$	50
$P_5$	$c, d, e$	100
$P_6$	$d, e, f, h$	200

**Definition 1 (The erasable itemset).** Let  $X \subseteq I$  be an itemset. The gain of itemset  $X$  is computed as:

$$g(X) = \sum_{\{P_k | X \cap P_k \text{Items} \neq \emptyset\}} P_k.Val \quad (1)$$

and  $X$  is an erasable itemset if:

$$g(X) \leq \mathcal{T} \times \xi \quad (2)$$

where  $\mathcal{T} = \sum_{P_k \in DB} P_k.Val$  is the total profit (for the factory) and  $\xi$  is a threshold.

The gain of itemset  $X$  is thus the sum of gains of the products which include at least one item in  $X$ . For the example dataset,  $\mathcal{T} = 1,700$  dollars. Let  $X = \{fh\}$  and  $\xi = 30\%$ . The gain of  $X$  is  $g(X) = P_4.Val + P_6.Val = 250$  dollars because  $P_4$  and  $P_6$  include  $\{f\}$  or  $\{h\}$  or  $\{fh\}$  as items.  $X$  is called an erasable itemset because  $g(X) = 250 < \mathcal{T} \times \xi = 1,700 \times 30\% = 510$  dollars.

In [2], Deng et al. presented the **WPPC-tree**, a tree of the form  $\langle N.item\text{-}name, N.weight, N.childnodes, N.pre, N.post \rangle$  (see Fig. 1). The root of the WPPC-tree,  $\mathcal{R}$ , has the  $\mathcal{R}.item\text{-}name = \text{"null"}$  and  $\mathcal{R}.weight = 0$ . The WPPC-tree is created by Algorithm 1, WPPC-tree construction, presented in [2]. For each node  $N$  in the WPPC-tree, the tuple  $\langle N.pre, N.post: N.weight \rangle$  is called the **WPP-code** of  $N$ . A WPP-code  $\langle N_i.pre, N_i.post: N_i.weight \rangle$  is an **ancestor** of another WPP-code  $\langle N_j.pre, N_j.post: N_j.weight \rangle$  if and only if  $N_i.pre < N_j.pre$  and  $N_i.post > N_j.post$ . For example, Fig. 1 shows the complete WPPC-tree of the dataset presented in Table 1 in which only include the erasable 1-itemset. Two in-erasable 1-itemsets  $a$  and  $b$  thus are not exist in this figure. In Fig. 1, the WPP-code of the highlighted node  $N_1$  is  $\langle 1,5:500 \rangle$  and the WPP-code of  $N_2$  is  $\langle 6,3:100 \rangle$ .  $N_1$  is an ancestor of  $N_2$  because  $N_1.pre = 1 < N_2.pre = 6$  and  $N_1.post = 5 > N_2.post = 3$ .

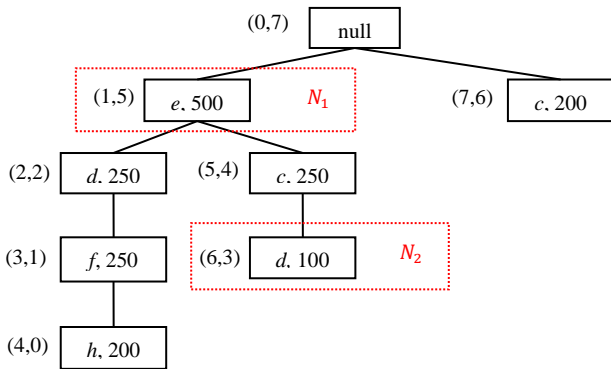


Figure 1. The WPPC-tree for the example dataset with  $\xi = 30\%$

From the WPPC-tree an **NC-Set** is a list of WPP-codes associated with an erasable 1-itemset. Note that the NC-Set of a 1-itemset is sorted in descending order of  $N_i.pre$  and the collection of erasable 1-itemsets  $E_1$  is sorted in descending order of frequency. For instance, the NC-Set of  $c$  is  $\{\langle 5,4:250 \rangle, \langle 7,6:200 \rangle\}$  and the NC-Set of  $d$  is  $\{\langle 2,2:250 \rangle, \langle 6,3:100 \rangle\}$ .

Assume that  $XA$  and  $XB$  are two erasable  $k$ -itemsets with the same prefix  $X$ .  $NC_1$  and  $NC_2$  are the NC-Sets of  $XA$  and  $XB$  respectively. The **NC-Set**  $NC_3$  of  $XAB$  ( $A > B$ :  $A$  is before  $B$  in  $E_1$ ) is determined by combining  $NC_1$  and  $NC_2$ . This combination is conducted according the following: (i) the WPP-codes in  $NC_3$  are sorted in ascending order of  $N_i.pre$ , (ii) all WPP-codes in  $NC_1$  are in  $NC_3$  and (iii) the WPP-codes in  $NC_2$  are in  $NC_3$  if no WPP-code in  $NC_1$  is its ancestor. For example, let  $XA = \{c\}$  and  $XB = \{d\}$ . The NC-Set of  $XAB = \{cd\}$  is  $\{\langle 2,2:250 \rangle, \langle 5,4:250 \rangle, \langle 7,6:200 \rangle\}$ .  $\langle 6,3:100 \rangle$  is excluded because  $\langle 5,4:250 \rangle$  is one of its ancestor.

The **gain** of  $k$ -itemset is then computed by summing the  $N_i.weights$  of WPP-codes in its NC-Set. For instance, the gain of  $\{cd\}$  is  $250 + 250 + 200 = 700$  dollars.

### III. THE DISCUSSIONS ON MERIT ALGORITHM

The MERIT algorithm for mining erasable itemsets proposed in [2] is an effective algorithm. However, this algorithm cannot find all erasable itemsets because of the following issues:

1. The **mining\_E** procedure [2] checks all the subset  $(k-1)$ -itemsets of a  $k$ -itemsets  $X$  are erasable or not to determine  $X$  be inerasable. However, MERIT uses the deep-first-search strategy so there are not enough the available  $(k-1)$ -itemsets in results (the temporary results in this step). Therefore, this check is always false where  $k > 2$  and all erasable  $k$ -itemsets ( $k > 2$ ) is always inerasable;
2. The **mining\_E** procedure [2] enlarges the equivalence classes of  $EC_v[k]$  therefore the results of MERIT does not include all erasable itemsets (although it does introduce efficiency gains).

To address the above issues we implemented an algorithm by removing the lines 7, 13, 15 and 16 in **mining\_E** procedure of MERIT, called MERIT+, to mining erasable itemsets fully.

Besides, MERIT algorithm featured a number of weaknesses:

1. It stores the weight value of each WPP-code which can appear in many erasable itemsets' NC-Sets, therefore, leading to much duplication;
2. It does not provide effective mechanisms for: (a) searching for a specific 1-itemset in an erasable 1-itemsets  $E_1$  and (b) determining whether an erasable 1-itemset is before or after other erasable 1-itemset;
3. It uses a strategy whereby an itemset  $X$ 's NC-Set is the subset of an itemset  $Y$ 's NC-Set if  $X \subset Y$ .

To overcome the above weaknesses of MERIT (and also MERIT+), dMERIT+ algorithm is proposed and described in the following section.

### IV. dMERIT+ ALGORITHM

This section introduces the dMERIT+ algorithm starting with some definitions.

**Definition 2 (The weight index).** Let  $\mathcal{R}$  be the WPPC-tree. We define  $W$ , a weight index:

$$W[N_i.pre] = N_i.weight \quad (3)$$

where  $N_i \in \mathcal{R}$  is the node in the WPPC-tree.

The weight index for the WPPC-tree given in Fig. 1 is presented in Table 2. Using the weight index, we propose a new WPP-code structure  $\langle N.pre, N.post \rangle$ , called **WPP'-code**, which helps the proposed algorithm reduce the required memory usage and easily determine the weight of the individual WPP'-code based on  $W$ .

TABLE II. THE WEIGHT INDEX  $W$  FOR THE EXAMPLE DATASET WITH  $\xi = 30\%$

Pre	0	1	2	3	4	5	6	7
Weight	0	500	250	250	200	250	100	200

**Definition 3 (The hash table of erasable 1-itemsets).** Let  $E_1$  be the erasable 1-itemsets. We define  $H_1$ , a hash table of erasable 1-itemsets.

$$H_1[e_i] = i \quad (4)$$

where  $e_i \in E_1$  is an erasable 1-itemset.

Once the hash table of erasable 1-itemsets has been generated, we can:

1. Determine whether a 1-itemset  $e_i$  is in  $E_1$  or not by considering whether this 1-itemset exists in  $H_1$  or not. If  $e_i$  exists in  $E_1$  dMERIT+ can also easily extract the information associated with  $e_i$  by “hashing” into the table to get the index of  $e_i$  in  $E_1$  (denoted by  $j$ ) and accessing the element at position  $j$  in  $E_1$ ;
2. Compare two erasable 1-itemsets  $e_1$  and  $e_2$  in the order of appearance in  $E_1$  by comparing  $H_1[e_1]$  and  $H_1[e_2]$ . If  $H_1[e_1] > H_1[e_2]$  means then  $e_2 > e_1$  and reverse.

Thus using the hash table reduces the runtime of dMERIT+ compared to MERIT+ (and consequently MERIT)

**Definition 4 (dNC-Set).** Let  $XA$  with its NC-Set  $NC(XA)$  and  $XB$  with its NC-Set  $NC(XB)$  be two itemsets with the same prefix  $X$ . Assume that  $NC^*(XB) = \{nc \in NC(XB) \mid \exists nc' \in NC(XA) \text{ so that } nc' \text{ is ancestor of } nc\}$ . The dNC-Set of  $NC(XA)$  and  $NC(XB)$  denoted by  $dNC(XAB)$  is defined as follow:

$$dNC(XAB) = NC(XB) \setminus NC^*(XB) \quad (5)$$

**Theorem 1:** Assume that we have  $g(XA)$ , the gain of  $XA$ . The gain of  $XAB$ ,  $g(XAB)$ , is computed as follow:

$$g(XAB) = g(XA) + \sum_{(pre,post) \in dNC(XAB)} W[pre] \quad (6)$$

where  $W[pre]$  is the element at position  $pre$  in  $W$ .

**Proof:** According to definition 7 in [2],  $NC(XAB) = NC(XA) \cup [NC(XB) \setminus NC^*(XB)] = NC(XA) \cup dNC(XAB)$ . So,  $g(XAB) = g(XA) + \sum_{(pre,post) \in dNC(XAB)} W[pre]$ , Theorem 1 is proved.

From the above, we propose the dMERIT+ algorithm as presented in Fig. 2.

**Input:** A product dataset  $DB$  and a threshold  $\xi$   
**Output:**  $E$ , the set of all erasable itemsets  
1. **Construct\_WPPC\_tree**( $DB, \xi$ ) to generate  $\mathcal{R}$ ,  $E_1$ ,  $H_1$  and  $\mathcal{T}$   
2. **GenerateNC-Sets**( $\mathcal{R}, E_1$ )  
3.  $E \leftarrow E_1$   
4. if  $E_1.size > 1$  then  
5. **mining\_E**( $E_1$ )  
6. return  $E$

Procedure **Construct\_WPPC\_tree**( $DB, \xi$ )  
1. scan  $DB$  once to find  $E_1$ , their gains, their frequency and the total gain of the factory( $\mathcal{T}$ )  
2. sort  $E_1$  in frequency descending order  
3. create  $H_1$ , the hash table of  $E_1$   
4. create the root of a WPPC-tree,  $\mathcal{R}$ , and label it as 'null'  
5. for each  $P \in DB$   
6. remove inerasable 1-itemsets  
7. sort its erasable 1-itemsets in frequency descending order  
8. **insert\_tree**( $P, \mathcal{R}$ )  
9. scan WPPC-tree to generate pre and post and  $H_1$   
10. return  $\mathcal{R}, E_1, H_1$  and  $\mathcal{T}$

Procedure **insert\_tree**( $P, \mathcal{R}$ )  
1. while ( $P$  is not null) do  
2.  $P_1 \leftarrow$  The first items of  $P$   
3.  $P \leftarrow P \setminus P_1$   
4. if  $\mathcal{R}$  has a child  $N$  such that  $N.item-name = P_1$  then  
5.  $N.Weight = N.Weight + P.Val$   
6. else  
7. create a new node  $N$   
8.  $N.Weight \leftarrow P.Val$   
9.  $\mathcal{R}.Childnodes = N$   
10. **insert\_tree**( $P, N$ )  
11. end while

Procedure **GenerateNC-Sets**( $\mathcal{R}, E_1$ )  
1.  $NC_1 \leftarrow \mathcal{R}.Post$  and  $\mathcal{R}.Pre$   
2.  $pos = H_1[\mathcal{R}.item-name]$   
3. add  $NC_1$  to  $E_1[pos].NC-Sets$  // by definition 3  
4. for each  $Child$  in  $\mathcal{R}.ChildNodes$   
5. **GenerateNC-Sets**( $Child$ )

Procedure **mining\_E**( $EC$ )

```

1. for  $k \leftarrow 1$  to  $EC.size$  do
2.    $EC_{next} \leftarrow \emptyset$ 
3.   for  $j \leftarrow (k+1)$  to  $EC.size$  do
4.     let  $e_1$  and  $e_2$  be the last item of
 $EC[k].Items$  and  $EC[j].Items$  respectively
5.     if  $H_1[e_1] < H_1[e_2]$  then // by definition 3
6.        $EI.Items \leftarrow EC[k].Items + \{e_2\}$ 
7.        $(EI.NC-Sets \text{ and } Gain) \leftarrow \mathbf{dNC-Set}(EC[k].NC-Sets, EC[j].NC-Sets)$ 
8.        $EI.Gain = EC[k].Gain + Gain$ 
9.     else
10.       $EI.Items \leftarrow EC[j].Items + \{e_1\}$ 
11.       $(EI.NC-Sets \text{ and } Gain) \leftarrow \mathbf{dNC-Set}(EC[j].NC-Sets, EC[k].NC-Sets)$ 
12.       $EI.Gain = EC[j].Gain + Gain$ 
13.      if  $(EI.Gain \leq \xi \times \mathcal{T})$ 
14.        add EI to  $EC_{next}$ 
15.        add EI to  $E$ 
16.      if  $EC_{next}.size > 1$  then
17.        mining_E( $EC_{next}$ )

```

Function **dNC-Set**( $NC_1, NC_2$ )

```

1.  $NC_3 \leftarrow \emptyset$  and  $Gain \leftarrow 0$ 
2. for each  $nc_2 \in NC_2$ 
3.    $flag = 0$ 
4.   for each  $nc_1 \in NC_1$ 
5.     if  $nc_1$  is ancestor of  $nc_2$  then  $flag = 1$  and
break
6.   if ( $flag == 0$ )
7.     insert  $nc_2$  to  $NC_3$ 
8.      $Gain = Gain + W[nc_2.Pre]$  // by theorem 1
and definition 2
9. return  $NC_3$  and  $Gain$ 

```

Figure 2. dMERIT+ algorithm

## V. THE EXAMPLE

Considering the example data set given in Table 1,  $\xi = 30\%$  and the dMERIT+ algorithm (Fig. 2). First, dMERIT+ calls the *Construct\_WPPC\_tree* procedure to create the WPPC\_tree  $\mathcal{R}$  (see Fig. 1), the erasable 1-itemsets  $E_1$ , the hash table of erasable 1-itemsets  $H_1$  (see Table. 2) and the total gain for the factory  $\mathcal{T}$ . The *GenerateNC-Sets* procedure was used to create  $E_1$ 's associated NC-Sets. The *mining\_E* procedure is then called with  $E_1$  as a parameter all so as to identify the complete set of erasable itemsets (Fig. 3).

We also ran MERIT+ on the example data set with  $\xi = 30\%$ , then we obtained the erasable itemsets and confirmed that both algorithms find the same set of erasable itemsets although in different ways.

When considering the memory usage associated with both algorithms the following can be observed:

1. The memory usage can be determined by summing either: (i) the memory to store erasable itemsets, their new NC-Sets, the set of WPP'-codes  $\langle N.pre, N.post \rangle$ , and the weight

index (dMERIT+) or (ii) the memory to store erasable itemsets and their NC-Sets (MERIT+);

2.  $N.pre, N.post, N.weight$ , the item identifier and the gain of an erasable itemset are represented in an integer format which requires 4 bytes in memory.

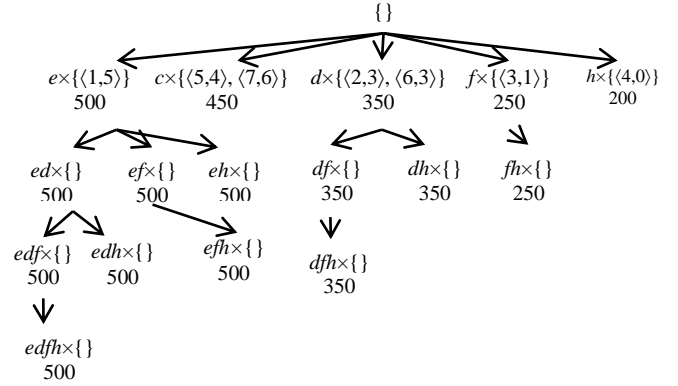


Figure 3. The complete set of erasable itemsets identified by applying dMERIT+ to the example data set with  $\xi = 30\%$

We count the number of items included in the dMERIT+'s output (see Fig. 3) which is 63. In addition, dMERIT+ also requires an array with 7 elements as the weight index. Therefore, the memory usage required by dMERIT+ is  $(63+7) \times 4 = 280$  bytes. Meanwhile, the erasable itemsets and their NC-Sets of MERIT+ algorithm require 112, the number of items. Hence, the memory usage required by MERIT+ is  $112 \times 4 = 448$  bytes.

Thus the example shows that the memory usage for dMERIT+ is less than the memory usage for MERIT+.

## VI. EXPERIMENTAL RESULTS

Both MERIT+ and dMERIT+ were encoded using C# 2012 on an ASUS laptop with Intel core i3-3110M 2.4GHz and 4GBs of RAM. We used the Mushroom, Pumsb and T10I4D100K datasets downloaded from <http://fimi.cs.helsinki.fi/data/> but adding a column indicating the "profit" for each product. The profit of each product was generated using a function  $Rand(100, 50)$ , where the mean value is 100 and the variance is 50. Some statistics concerning these datasets are shown in Table 3.

TABLE III. THE FEATURES OF THE PRODUCT DATASETS

Dataset	#Products	#Items
Mushroom	8,124	120
Pumsb	49,046	7,117
T10I4D100K	100,000	870

The objectives of the experiments were to compare the runtime and the memory usage of both MERIT+ (described in Section 3) and dMERIT+ (described Section 4). Note that:

1. Runtime in this context is the period between the start of the input to the end of the output.

2. Memory usage was determined as described at the end of Section 5.

Figs. 4 to 6 show the recorded runtimes when MERIT+ and dMERIT+ were applied to the three datasets. From the tables it can be seen that dMERIT+ outperforms MERIT+ in terms of the runtime, especially with respect to datasets which have a large number of items (Fig. 4 and Fig. 6).

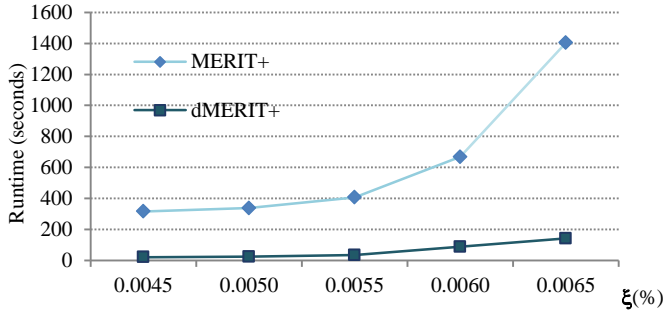


Figure 4. The runtime of MERIT+ and dMERIT+ on Pumsb dataset

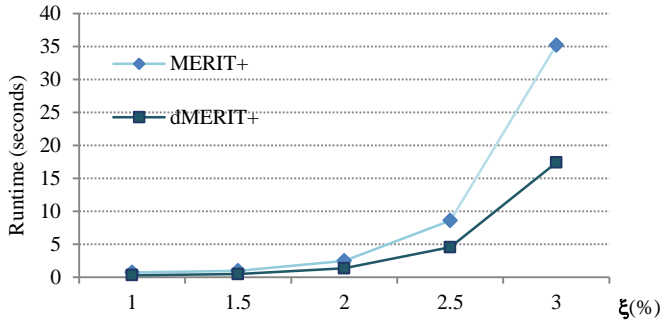


Figure 5. The runtime of MERIT+ and dMERIT+ on Mushroom dataset

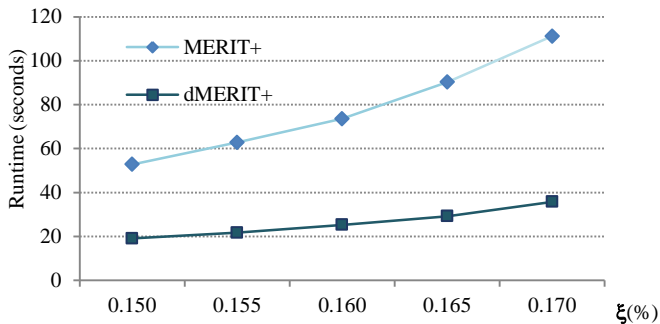


Figure 6. The runtime of MERIT+ and dMERIT+ on T1014D100K dataset

## VII. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented MERIT+, a revised version of MERIT algorithm for mining erasable itemsets fully. We then proposed dMERIT+, the main contribution of this paper, an advanced algorithm that uses the weight index, the hash table

of erasable 1-itemsets and the dNC-Sets to reduce the runtime and memory usage associated with both MERIT and MERIT+. The experimental results demonstrated that dMERIT+ is more efficient than MERIT+ (and naturally MERIT).

For future work we will initially focus on mining erasable closed itemsets. In the longer term, inspired by work on, mining frequent itemsets in incremental datasets [8-11], we study how to mine erasable itemsets in this setting.

## REFERENCES

- [1] Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. VLDB'94, 487-499, 1994.
- [2] Deng, Z.H., Xu, X.R.: Fast mining erasable itemsets using NC\_sets. Expert Systems with Applications 39(4), 4453-4463, 2012.
- [3] Deng, Z.H., Xu, X.R.: An efficient algorithm for mining erasable itemsets. ADMA'10, 214-225, 2010.
- [4] Deng, Z., Fang, G., Wang, Z., Xu, X.: Mining erasable itemsets. ICMLC'09, 67-73, 2009.
- [5] Dong, J., Han, M.: BitTable-FI: An efficient mining frequent itemsets algorithm. Knowledge Based Systems, 20(4), 329-335, 2007.
- [6] Grahne, G., Zhu, J.: Fast algorithms for frequent itemset mining using fp-trees. IEEE Transactions on Knowledge and Data Engineering, 17(10), 1347-1362, 2005.
- [7] Han J., Pei J., Yin Y.: Mining frequent patterns without candidate generation. SIGMOD'00, 1-12, 2000.
- [8] Hong, T.P., Wang, C.Y., Tao, Y.H.: A new incremental data mining algorithm using pre-large itemsets. Intelligent Data Analysis 5(2), 111-129, 2001.
- [9] Le, T.P., Hong, T.P., Vo, B., Le, B.: Incremental mining frequent itemsets based on the trie structure and the prelarge itemsets. GRC'11, 369-373, 2011.
- [10] Le, T.P., Vo, B., Hong, T.P., Le, B.: An efficient incremental mining approach based on IT-tree. RIVF'12, 57-61, 2012.
- [11] Lin, C.W., Hong, T.P., Lu, W.H.: Using the structure of prelarge trees to incrementally mine frequent itemsets. New Generation Computing 28(1), 5-20, 2010.
- [12] Pei, J., Han, J., Mao, R.: CLOSET: An efficient algorithm for mining frequent closed itemsets. SIGMOD workshop on research in data mining and knowledge discovery, 11-20, 2000.
- [13] Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Discovering Frequent Closed Itemsets for Association Rules. ICDT'12, 398 - 416, 1999.
- [14] Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Efficient Mining of Association Rules using Closed Itemset Lattices. Information Systems 24(1), 25 - 46, 1999.
- [15] Song, W., Yang, B., Xu, Z.: Index-BitTableFI: An improved algorithm for mining frequent itemsets. Knowledge Based Systems, 21(6), 507-513, 2008.
- [16] Vo, B., Hong, T.P., Le, B.: Dynamic bit vectors: An efficient approach for mining frequent itemsets. Scientific Research and Essays, 6(25), 5358-5368, 2011.
- [17] Vo, B., Hong, T.P., Le, B.: DBV-Miner: A dynamic bit-vector approach for fast mining frequent closed itemsets. Expert Systems with Applications 39(8), 7196-7206, 2012.
- [18] Zaki, M.J., Hsiao, C.J.: Efficient algorithms for mining closed itemsets and their lattice structure. IEEE Transactions on Knowledge and Data Engineering, 17(4), 462-478, 2005.
- [19] Wang, J., Han, J., Pei, J.: CLOSET+: Searching for the best strategies for mining frequent closed itemsets. SIGKDD'03, 236-245, 2003.