An Efficient Algorithm for Mining Erasable Itemsets

Zhihong Deng[1,2] and Xiaoran Xu[1]

[1] Key Laboratory of Machine Perception (Ministry of Education),
School of Electronics Engineering and Computer Science,
Peking University, Beijing 100871, China
[2] The State Key Lab of Computer Science, Institute of Software,
Chinese Academy of Sciences, Beijing 100190, China
zhdeng@cis.pku.edu.cn, xuxiaoran@pku.edu.cn

Abstract. Mining erasable itemsets first introduced in 2009 is one of new emerging data mining tasks. In this paper, we present a new data representation called PID_list, which keeps track of the id_nums (identification number) of products that include an itemset. Based on PID_list, we propose a new algorithm called VME for mining erasable itemsets efficiently. The main advantage of VME algorithm is that the gain of an itemset can be computed efficiently via union operations on product id_nums. In addition, VME algorithm can also automatically prune irrelevant data. For evaluating VME algorithm, we have conducted experiments on six synthetic product databases. Our performance study shows that the VME algorithm is efficient and is on average over two orders of magnitude faster than the META algorithm, which is the first algorithm for dealing with the problem of erasable itemsets mining.

Keywords: Data mining, Erasable Itemsets, Algorithm, PID_list.

1 Introduction

Since the problem of mining frequent patterns first introduced in [2], it has emerged as a fundamental problem in data mining and plays an essential role in many important data mining tasks such as association rule analysis, cluster analysis, classification, and many other important data mining tasks [2]. Although there are plenty of studies on pattern mining, such as in [3], many new pattern-mining problems have arisen, such as high-utility pattern mining [4], probabilistic frequent itemsets mining [5], erasable itemsets mining [6], and so on, with the extensive application of pattern mining in every walk of life.

The problem of mining erasable itemsets originates from production planning. Consider a manufacturing factory, which produces a large collection of products. Each type of product is made up of a few components (or materials). For manufacturing their products, the factory should spend a large number of money to purchase or store these components. When financial crisis is coming, the factory should carefully plan production because it has not enough money to purchase all needed components as usual. Therefore, a vital question to the managers of the factory is how to plan the manufacture of production due to limited money. They can not purchase all components due to limited money. Obviously, they must stop manufacturing some productions because the corresponding components are unavailable. However, for the sake of commercial interests, the loss of the factory's profit caused by stopping manufacturing some products should be controllable. Hence, the key to the problem is how to efficiently find these components, without which the loss of the profit is no more than the given threshold. These components are also called as erasable itemsets. The paper [6] first introduced the problem of erasable itemsets mining and proposed META algorithm to deal with the problem.

Although META algorithm is capable of finding all erasable itemsets in reasonable time, it has two important weaknesses. The first weakness is that the time efficiency of META algorithm is poor because it scans database repeatedly. The second weakness is that META algorithm can not automatically prune irrelevant data.

In this paper, we present a new algorithm, which can overcome the weaknesses of META, to mine erasable itemsets efficiently. First, we present a new data representation called PID_list, which keeps track of the id_nums of products that include an itemset. Second, we propose a PID_list-based algorithm called VME for mining erasable itemsets efficiently. The main advantage of VME algorithm is that the gain of an itemset can be computed efficiently via union operations on product id_nums. In addition, VME algorithm can also automatically prune irrelevant data. For evaluating VME algorithm, we have conducted experiments on six synthetic product databases. Our performance study shows that the VME algorithm is efficient and is on average over two orders of magnitude faster than the META algorithm, which is the first algorithm for dealing with the problem of erasable itemsets mining.

The organization of the rest of the paper is as follows. Section 2 introduces the formal statement of the problem. Section 3 introduces the PID_list structure and its properties. Section 4 develops a PID_list-based erasable itemsets mining algorithm, VME. Section 5 presents our performance study. In section 6, we conclude with a summary and point out some future research issues.

2 Problem Statement and Preliminaries

Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of items, which are the abstract representation of components of products, and a product database $DB = \{P_1, P_2, \ldots, P_n\}$, where P_i ($i \in [1\ldots n]$) is a type of product and is presented in the form of $< PID, Items, Val >$. *PID* is the identifier of P_i. *Items* are all items (or components) that constitute P_i. *Val* is the profit that a manufactory (or factory) reaps (or obtains) by selling all P_i-type products. In this section, related concepts are described and a formal description of erasable itemset is given.

Table 1 shows an example product database. The database has six types of product. $\{i_1, i_2, i_3, i_4, i_5, i_6, i_7\}$ is the set of universal items that comprise the six types of products .Let's take P_3 as an example. We know that we need four different kind of item, which are i_1, i_2, i_3 and i_5, to manufacture P_3-type products. It would profit 50 million dollars by selling all P_3-type products.

Table 1. Example product database

Product	***PID***	***Items***	***Val* (Million $)**
P_1	1	$\{i_2, i_3, i_4, i_6\}$	50
P_2	2	$\{i_2, i_5, i_7\}$	20
P_3	3	$\{i_1, i_2, i_3, i_5\}$	50
P_4	4	$\{i_1, i_2, i_4\}$	800
P_5	5	$\{i_6, i_7\}$	30
P_6	6	$\{i_3, i_4\}$	50

Definition 2.1: Let A ($\subseteq I$) be an itemset (a set of items), the gain of A is defined as:

$$Gain(A) = \sum_{\{P_k | A \cap P_k.Items \neq \phi\}} P_k.Val\,. \tag{1}$$

That is, the gain of itemset A is the sum of profits of all products that include at least one item in A as their components.

Let P (=$\{i_6, i_7\}$) be an itemset. From table 1, we know that the products, who's *Items* contain i_6 or i_7, are P_1, P_2 and P_5. Therefore, the gain of P is the sum of $P_1.Val$, $P_2.Val$, and $P_5.Val$. That is, $Gain(P)$ is 100 million dollars.

Definition 2.2: Given a predefined threshold ξ and a product database DB, an itemset A is erasable if

$$Gain(A) \leq (\sum_{P_k \in DB} P_k.Val) \times \xi\,. \tag{2}$$

Based on the above definitions, the problem of mining erasable itemsets can be described as follows:

Given a product database, DB, and a threshold, ξ, the problem of finding the complete set of erasable itemsets is called the erasable itemsets mining problem.

Erasable itemsets are those itemsets that without these items, the loss of profits does not exceed ξ percents of the original profits. For example, let ξ be 10%. $\{i_6, i_7\}$ is an erasable. If a manufactory does not purchase i_6 and i_7 as raw materials because of economic crisis, the manufactory can not manufacture products that are type of P_1, P_2 or P_5. However, the lost profit is no more than 10% of the original profit. Erasable itemsets is especially useful for manufactures to decide how to purchase raw materials and plan the process of manufacturing products in the case of economic crisis.

Consider the example product database shown in Table 1. Let ξ be 15%. According to Definition 2, the entire erasable itemsets are $\{i_3\}$ (150), $\{i_5\}$ (70), $\{i_6\}$ (80), $\{i_7\}$ (50), $\{i_5, i_6\}$ (150), $\{i_5, i_7\}$ (100), $\{i_6, i_7\}$ (100), and $\{i_5, i_6, i_7\}$ (150). Note that the numbers in parentheses are the gains of corresponding itemsets.

3 PID_list: Definition and Property

Obviously, the key question of erasable itemset mining is how to compute the gain of an itemset. META algorithm employs the method of scanning database to get the gains. However, after some careful examination, we find that there exist more efficient methods if we change the date format of product databases.

To design an efficient data structure for fast mining, let's first examine the database showed by Table 2. By careful observation we find the database in Table 2 is an inversion of the database in Table 1. The data format employed by the database in Table 1 is known as horizontal data format while the data format employed by the database in Table 2 is known as vertical data format.

Table 2. The inversion database

Item	*Inverted List*
i_1	<3, 50>, <4, 800>
i_2	<1, 50>, <2, 20>, <3, 50>, <4, 800>
i_3	<1, 50>, <3, 50>, <6, 50>
i_4	<1, 50>, <4, 800>, <6, 50>
i_5	<2, 20>, <3, 50>
i_6	<1, 50>, <5, 30>
i_7	<2, 20>, <5, 30>

One advantage of vertical data format is that we can very convenient to obtain the gain of itemsets. For example, the gain of $\{i_3\}$ is the sum of the second part of <1, 50>, <3, 50>, and <6, 50>. That is, it is 150 (50+50+50). In fact, not only 1-itemsets but also k-itemset ($k > 1$) can also be easy computed by a similar method, where a k-itemset means an itemset including k items. For example, by combing the *Inverted List* of $\{i_5\}$, {<2, 20>, <3, 50>}, and the *Inverted List* of $\{i_6\}$, {<1, 50>, <5, 30>}, we generate {<1, 50>, <2, 20>, <3, 50>, <5, 30>}. By summing 50, 20, 50, and 30, we get 150, which is the gain of $\{i_5, i_6\}$. As for the rationality, the remainder of this section will be discussed in detail.

The other advantage of vertical data format is that it can automatically prune irrelevant data. Let's see $\{i_3\}$ again. If we want to get the gain of $\{i_3\}$ by database scanning, we must compare it with each product of the database in Table 1 to examine whether it is a component of the product. Obviously, comparing $\{i_3\}$ with P_1, P_2 and P_5 is nonsense because they do not take i_3 as their component at all. If we do this by searching the *Inverted List* of $\{i_5\}$, P_1, P_2 and P_5 are filtered automatically because they are not in the *Inverted List* of $\{i_5\}$.

Before we introduce PID_list, we give two conventions in the paper for the sake of discussion. First, we assume that $I = \{i_1, i_2, \ldots, i_m\}$ is the given universal item set and $DB = \{P_1, P_2, \ldots, P_n\}$ is the given product database. Second, we denote an itemset $X(\subseteq I)$ by $\{i_{x1}, i_{x2}, \ldots, i_{xk}\}$, where $i_{xj} \in I$ for $1 \le j \le k$ and $x_s < x_t$ for $1 \le s < t \le k$. That is, any itemset is an ordered set and is sorted by the order that items occur in I.

Definition 3.1: For any $y \in I$, the PID_list of 1-itemset $\{y\}$ is $\{<P_{y1}.PID, P_{y1}.Val>, <P_{y2}.PID, P_{y2}.Val>, \ldots, <P_{yk}.PID, P_{yk}.Val>\}$, where *y is a component of* P_{yj} for $1 \le j \le k$ (that is, $y \in P_{yj}.Items$) and $P_{yv}.PID$ is less than $P_{yu}.PID$ for $v < u$. that is, the elements in the PID_list are sorted by the ascending order of *PID*.

For example, the PID_list of $\{i_2\}$ is <1, 50>, <2, 20>, <3, 50>, <4, 800>.
According to the definition 2.1 and definition 3.1, we have the following Property 3.1.

Property 3.1: Let $\{y\}$ be a 1-itemset and its PID_list is $\{< P_{y1}.PID, P_{y1}.Val >, < P_{y2}.PID, P_{y2}.Val >, \ldots, < P_{yk}.PID, P_{yk}.Val >\}$. The gain of $\{y\}$ can be computed as follows:

$$Gain(\{y\}) = \sum_{j=1}^{k} P_{yj}.Val \ . \tag{3}$$

For example, the gain of $\{i_2\}$ is 920.

Based on Definition 3.1, we define the PID_list of a k-item (k >1) as follows.

Definition 3.2: Let $X = \{i_{x1}, i_{x2}, \ldots, i_{xk}\}$ be a k-itemset. For each i_{xj} ($1 \le j \le k$), we denote the PID_list of $\{i_{xj}\}$ as PID_list ($\{i_{xj}\}$). PID_list(X), the PID_list of X, is $\{< P_{X1}.PID, P_{X1}.Val >, < P_{X2}.PID, P_{X2}.Val >, \ldots, < P_{Xs}.PID, P_{Xs}.Val >\}$, which meets the following three conditions:

(1) $P_{xv}.PID$ is less than $P_{xu}.PID$ for $1 \le v < u \le s$;
(2) $\forall < P_{Xj}.PID, P_{Xj}.Val >$ $(1 \le j \le s) \in$ PID_list(X) $\Rightarrow \exists\ i_{xv} \in X$, $< P_{Xj}.PID, P_{Xj}.Val > \in$ PID_list ($\{i_{xv}\}$);
(3) For any $i_{xv} \in X$ $(1 \le v \le k)$, $\forall < P_j.PID, P_j.Val > \in$ PID_list ($\{i_{xv}\}$) $\Rightarrow < P_j.PID, P_j.Val > \in$ PID_list(X).

In fact, PID_list(X) is the union of the PID_lists of all items that belongs to X with the condition that its elements are sorted by the ascending order of *PID*.

Let's see an example. From Table 2, we know the PID_lists of $\{i_3\}$, $\{i_5\}$, $\{i_6\}$ are {<1, 50>, <3, 50>, <6, 50>}, {<2, 20>, <3, 50>}, and {<1, 50>, <5, 30>}. According to Definition 4, the PID_list of $\{i_3, i_5, i_6\}$ is {<1, 50>, <2, 20>, <3, 50>, <5, 30>, <6, 50>}.

If we sum the second part of each element in the PID_list of $\{i_3, i_5, i_6\}$, we would find that it is 200. Surprisingly, the gain of $\{i_3, i_5, i_6\}$ is also 200. This is not an accidental phenomenon. In fact, we have the following Property 3.2.

Property 3.2. Let X be a itemset and its PID_list is $\{< P_{X1}.PID, P_{X1}.Val >, < P_{X2}.PID, P_{X2}.Val >, \ldots, < P_{Xs}.PID, P_{Xs}.Val >\}$. The gain of X can be computed as follows:

$$Gain(X) = \sum_{j=1}^{s} P_{Xj}.Val \ . \tag{4}$$

Proof. If X be 1-itemset, we know equation (4) is right according to Property 3.1.Then, we see the case that X is a k-itemset (k > 1). According to Definition 2.1, the gain of X is the sum of profits of all products that include at least one item in X as their components. The set of these products can be formally represented by $\{P_k \mid X \cap P_k.Items \neq \varnothing\}$. The products that occur in PID_list of X can be formally represented by $\{P_{X1}, P_{X2}, \ldots, P_{Xs}\}$. So, we convert the proof of equation (4) to the proof of $\{P_k \mid X \cap P_k.Items \neq \varnothing\} = \{P_{X1}, P_{X2}, \ldots, P_{Xs}\}$. We split the proof into two steps as follows.

(***a***) $\{P_k \mid X \cap P_k.Items \neq \varnothing\} \subseteq \{P_{X1}, P_{X2}, \ldots, P_{Xs}\}$

For any $P \in \{P_k \mid X \cap P_k.Items \neq \varnothing\}$, we know that there must exist i, $i \in X \cap P_k.Items$. According to the Definition 3.1, <$P.PID$, $P.Val$> must be an element of

PID_list ($\{i\}$) because of $i \in P_k.Items$. According to the condition (3) of Definition 3.2, we have $<P.PID, P.Val> \in$ PID_list(X). That is, $P \in \{P_{X1}, P_{X2}, \ldots, P_{Xs}\}$. So, we have $\{P_k \mid X \cap P_k.Items \neq \varnothing\} \subseteq \{P_{X1}, P_{X2}, \ldots, P_{Xs}\}$.

(***b***) $\{P_{X1}, P_{X2}, \ldots, P_{Xs}\} \subseteq \{P_k \mid X \cap P_k.Items \neq \varnothing\}$

For any $P \in \{P_{X1}, P_{X2}, \ldots, P_{Xs}\}$, we know $<P.PID, P.Val> \in$ PID_list(X) according to the definition of $\{P_{X1}, P_{X2}, \ldots, P_{Xs}\}$. According to the condition (2) of Definition 3.2, we know $\exists\, i \in X$, $<P.PID, P.Val> \in$ PID_list ($\{i\}$). According to Definition 3.1, we have $i \in P.Items$. So, we have $(i \in X) \wedge (i \in P.Items)$. That is, $P \in \{P_k \mid X \cap P_k.Items \neq \varnothing\}$. Therefore, we have $\{P_{X1}, P_{X2}, \ldots, P_{Xs}\} \subseteq \{P_k \mid X \cap P_k.Items \neq \varnothing\}$.

Consolidating (***a***) and (***b***), we prove that equation (4) is right when X is a k-itemset ($k > 1$).

Based on the above analysis, we prove Property 3.2. □

4 Mining Erasable Itemsets Using PID_list

Before presenting our method, let's first explore the following lemmas relevant to erasable itemsets mining.

Lemma 4.1: Let X ($\subseteq I$) and Y ($\subseteq I$) be two itemsets. If Y is a superset of X ($X \subseteq Y$), we have $Gain(X) \leq Gain(Y)$.

Proof. Let P_k be any product that satisfies $P_k.Items \cap X \neq \varnothing$. That is, $P_k.Items$ and X share at least one item. Because Y is X's superset, we have $P_k.Items \cap Y \neq \varnothing$. So, we have $\{P_k \mid P_k.Items \cap X \neq \varnothing\} \subseteq \{P_k \mid P_k.Items \cap Y \neq \varnothing\}$. According to Definition 2.1, we know

$$\sum_{\{P_k \mid X \cap P_k.Items \neq \phi\}} P_k.Val \leq \sum_{\{P_k \mid Y \cap P_k.Items \neq \phi\}} P_k.Val \,. \tag{5}$$

This means $Gain(X) \leq Gain(Y)$. □

Let's take Table 1 as an example. The values of itemset $\{i_6\}$, $\{i_6, i_7\}$, $\{i_3, i_6, i_7\}$ are 80, 100, and 200 respectively. It is obvious that $Val(\{i_6\}) \leq Val(\{i_6, i_7\}) \leq Val(\{i_3, i_6, i_7\})$.

Based on Lemma 4.1, we have Lemma 4.2 as follows.

Lemma 4.2 (anti-monotone): if itemset X is unerasable and Y is a superset of X ($X \subseteq Y$), Y must also be unerasable.

Proof. Let Y be a superset of X. We assume Y is erasable. According to Definition 2.2, we have

$$Gain(Y) \leq \Big(\sum_{P_k \in DB} P_k.Val\Big) \times \xi \cdot \tag{6}$$

Because X is unerasable, we have

$$Gain(X) > \sum_{P_k \in DB} P_k.Val \times \xi \cdot \tag{7}$$

Based on inequation (6) and inequation (7), we know *Gain*(X) > *Gain*(Y). However, according to Lemma 4.1, we know *Gain*(X) ≤ *Gain*(Y). These two conclusions conflict each other. Therefore, our assumption is wrong and Y must be unerasable. □

Based on Lemma 4.2, we employ an iterative approach known as a *level-wise* search, which is also adopted by Apriori algorithm [3] in frequent patterns mining. The *level-wise*-based iterative approach finds erasable (k+1)-itemsets by taking use of erasable k-itemsets.

The detailed idea of the *level-wise*-based iterative approach is described as follows. First, the set of erasable 1-itemsets is found. This set is denoted as E_1. E_1 is used to find E_2, which is the set of erasable 2-itemsets, and then E_2 is used to find E_3, and so on, until no more erasable k-itemsets can be found. The finding of each E_j requires one scan of the database. To improve the efficiency of the level-wise generation of erasable itemsets, Lemma 4.2 is used to narrow the range of search.

Based on the above discussions, we design an algorithm called VME algorithm. VME is the abbreviation for **V**ertical-format-based algorithm for **M**ining **E**rasable itemsets. The details of the VME algorithm are described as follows.

```
Algorithm: VME
Input: a product database, $DB$, a itemsets, $I$, and a
threshold, $\xi$.
Output: $EI$, all erasable itemsets in $DB$.
Method:
Scan $DB$ to get the overall profit, $Sum\_val$;
Scan $DB$ again to find the set of all erasable 1-
itemset, $E_1$, and the PID_list of each erasable 1-
itemset;
For ($k$ = 2; $E_{k-1} \neq \emptyset$; $k$++) {
  $GC_k$ = Gen_Candidate($E_{k-1}$);
  $E_k = \emptyset$;
  For each $k$-itemset $P \in GC_k$ {
    Compute $P.gain$ according to Property 3.2;
    If $P.gain \leq \xi \times Sum\_val$ then $E_k = E_k \cup \{P\}$;}}
Return $EI = \cup_k E_k$;
// Generating candidate itemsets and their PID_lists
Procedure Gen_Candidate($E_{k-1}$)
Candidates = $\emptyset$;
For each erasable itemset $A_1$(={$x_1$, $x_2$ , …$x_{k-2}$, $x_{k-1}$})$\in E_{k-1}$ {
  For each erasable itemset $A_2$(={$y_1$, $y_2$ , …$y_{k-2}$, $y_{k-1}$})$\in E_{k-1}$
{
    //Here, $x_{k-1} < y_{k-1}$ means $x_{k-1}$ is ahead of $y_{k-1}$ in $I$ .
    If $((x_1 = y_1) \wedge (x_2 = y_2) \wedge \ldots \wedge (x_{k-2} = y_{k-2}) \wedge (x_{k-1} < y_{k-1}))$ then
{
      $X = \{x_1, x_2, \ldots x_{k-2}, x_{k-1}, y_{k-1}\}$;
      If No_Unerasable_Subset($X$, $E_{k-1}$) then {
         $X$. PID_list = $A_1$.PID_list $\cup$ $A_2$.PID_list;
         Candidates = Candidates $\cup$ {($X$, $X$. PID_list)}; }
      } } }
Return Candidates;

Procedure **No_Unerasable_Subset**($X$, $E_{k-1}$).
// $X$: a candidate $k$-itemset
//$E_{k-1}$: the set of all erasable ($k$ -1)-itemsets
For each ($k$ -1)-subset $X_s$ of $X$ {
   If $X_s \notin E_{k-1}$ then Return FALSE; }
Return TRUE;
```

5 Experimental Evaluation

In this section, we present a performance comparison of VME with the first erasable itemsets mining algorithms META algorithm. All the experiments were performed on an IBM xSeries 366 Server with 2G Memory. The operating system was Microsoft Windows 2000 Server. All the programs were coded in MS/Visual C++.

For experimental databases, we first generated three databases by IBM generator[1]. These databases are denoted by T15I10D100K, T20I10D100K, and T30I25D100K. Each of the three databases has 100, 000 tuples (or products) and 200 different items. The average production sizes of T15I10D100K, T20I10D100K, and T30I25D100K are 15, 20, and 30 respectively.

However, T15I10D100K, T20I10D100K, and T30I25D100K have no attribute (or column) for representing profit of products. To make these databases more like product databases, we add a new attribute (column or field), which is used to store the profit of a product, for each database. We employ two probability distributions, U(1, 100) and N(50, 25), to generate products' profit. U(1, 100) is an uniform distribution with the range of values of [1, 100]. N(50, 25) is a normal distribution with mean of 50 and variance of 25. Therefore, we have six databases by combining T15I10D100K, T20I10D100K, and T30I25D100K with U(1, 100) and N(50, 25). Table 3 shows the details of these databases.

[1] http://www.almaden. ibm.com/cs/quest/syndata.html

Table 3. The summary of the database

Database	*#Product*	*#Items*	*Probability Distribution of Profits*
T15U1_100	100,000	200	$U(1, 100)$
T15N50_25	100,000	200	$N(50, 25)$
T20U1_100	100,000	200	$U(1, 100)$
T20N50_25	100,000	200	$N(50, 25)$
T30U1_100	100,000	200	$U(1, 100)$
T30N50_25	100,000	200	$N(50, 25)$

The scalability of VME and META on T15U1_100 and T15N50_25 as the threshold increases from 1% to 7% is shown in Figure 1 and Figure 2 respectively. Figure 3 and Figure 4 respectively show the scalability of VME and META on T20U1_100 and T20N50_25 as the threshold decreases from 2% to 10%. Figure 5 and Figure 6 respectively show the scalability of VME and META on T30U1_100 and T30N50_25 as the threshold decreases from 3% to 15%. As shown in Figure 1 – 6, VME scales much better than META. Not matter which database is used, VME algorithm is always over two orders of magnitude faster than META algorithm on average. Especially, the bigger the threshold is, the more distinct the advantage of VME over META is. The efficiency of VME can be explained by two main factors: (1) computing the gain of an itemset via PID_lists is much efficient; (2) PID_lists also provide a natural way to prune irrelevant data automatically.

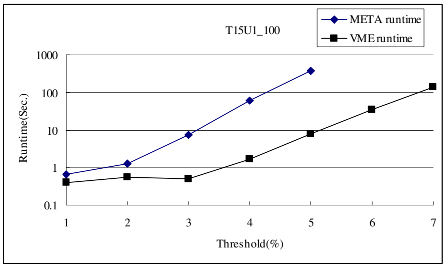

Fig. 1. Comparative performance on T15U1_100

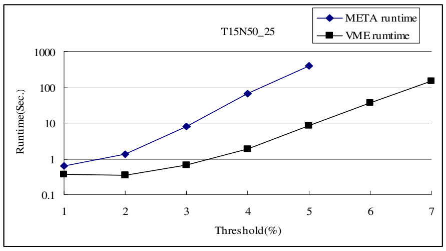

Fig. 2. Comparative performance on T15N50_25

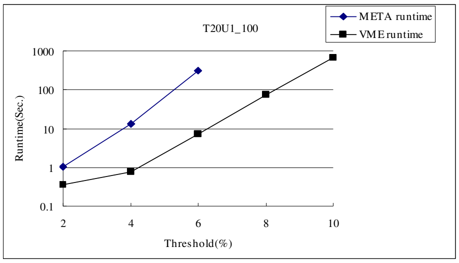

Fig. 3. Comparative performance on T20U1_100

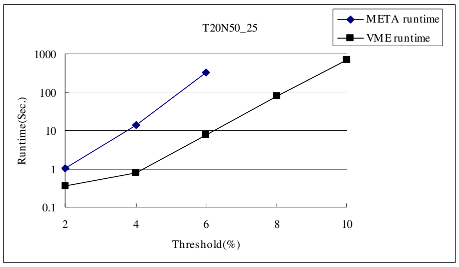

Fig. 4. Comparative performance on T20N50_25

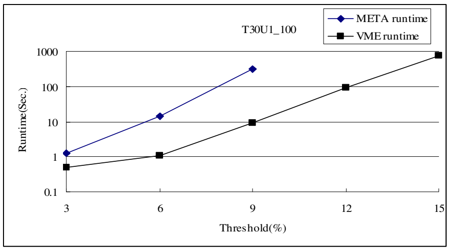

Fig. 5. Comparative performance on T30U1_100

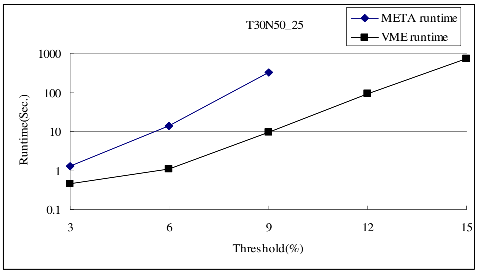

Fig. 6. Comparative performance on T30N50_25

6 Conclusions

In this paper, we present a new data representation called PID_list for storing compressed, crucial information about itemsets of a production database. Based on PID_list, we develop a new algorithm called VME for mining erasable itemsets efficiently. VME algorithm not only makes it easy to compute the gain of an itemset via union operations on product id_nums, but also automatically prune irrelevant data. For evaluating VME algorithm, we have conducted experiments on six synthetic product databases. Our performance study shows that the VME algorithm is efficient and is on average over two orders of magnitude faster than the META algorithm, the first algorithm for mining erasable itemsets.

For the future work, there is a lot of interesting research issues related to erasable itemsets mining. First, we will take efforts towards more efficient algorithms by adopting useful ideas from many proposed algorithms of mining frequent patterns. Second, there have been some interesting studies at mining maximal frequent [7], closed frequent patterns [8] and top-k frequent patterns [9, 10] in recent years. Similar to frequent patterns, the extension of erasable itemsets to these special forms is an interesting topic for future research.

Acknowledgments. This work is partially supported by the National High Technology Research and Development Program of China (863 Program) under Grant No. 2009AA01Z136 and the National Natural Science Foundation of China under Grant No.90812001. The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

References

1. Agrawal, R., Imielinski, T., Swami, A.: Mining Association Rules Between Sets of Items in Large Databases. In: Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, pp. 207–216. ACM Press, New York (1993)
2. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, pp. 1–12. ACM Press, New York (2000)
3. Han, J., Kamber, M.: Data Mining: Concepts and Techniques, 2nd edn. Morgan Kaufmann, San Francisco (2005)
4. Hu, J., Mojsilovic, A.: High-utility pattern mining: A method for discovery of High-utility item sets. Pattern Recognition 40, 3317–3324 (2007)
5. Bernecker, T., Kriegel, H., Renz, M., Verhein, F., Zuefle, A.: Probabilistic Frequent Itemset Mining in Uncertain Databases. In: 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 119–127. ACM Press, New York (2009)
6. Deng, Z., Fang, G., Wang, Z., Xu, X.: Mining Erasable Itemsets. In: 8th IEEE International Conference on Machine Learning and Cybernetics, pp. 67–73. IEEE Press, New York (2009)
7. Burdick, D., Calimlim, M.,, Flannick, J., Gehrke, J., Yiu, T.: MAFIA: A Maximal Frequent Itemset Algorithm. IEEE Trans. Knowledge and Data Engineering 17, 1490–1504 (2005)

8. Wang, J., Han, J., Pei, J.: CLOSET+: Searching for the Best Strategies for Mining Frequent Closed Patterns. In: 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 236–245. ACM Press, New York (2003)
9. Han, J., Wang, J., Lu, Y., Tzvetkov, P.: Mining Top-k Frequent Closed Patterns without Minimum Support. In: Second IEEE International Conference on Data Mining, pp. 211–218. IEEE Press, New York (2002)
10. Wang, J., Han, J., Lu, Y., Tzvetkov, P.: TFP: An Efficient Algorithm for Mining Top-k Frequent Closed Patterns. IEEE Trans. Knowledge and Data Engineering 17, 652–664 (2005)