Fast mining erasable itemsets using NC_sets

Zhi-Hong Deng *, Xiao-Ran Xu

Key Laboratory of Machine Perception (Ministry of Education), School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China

A R T I C L E   I N F O

Keywords:
Data mining
Erasable itemsets
NC_sets
Algorithms
Data structure

A B S T R A C T

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 NC_set, which keeps track of the complete information used for mining erasable itemsets. Based on NC_set, we propose a new algorithm called MERIT for mining erasable itemsets efficiently. The efficiency of MERIT is achieved with three techniques as follows. First, the NC_set is a compact structure, which prunes irrelevant data automatically. Second, the computation of the gain of an itemset is transformed into the combination of NC_sets, which can be completed in linear time complexity by an ingenious strategy. Third, MERIT can directly find erasable itemsets without generating candidate itemsets in some cases. For evaluating MERIT, we have conducted extensive experiments on a lot of synthetic product databases. Our performance study shows that the MERIT is efficient and is on average about two orders of magnitude faster than the META, the first algorithm for mining erasable itemsets.

1. Introduction

Data mining has attracted a great deal of attention in the information industry and in society as a whole in recent years, due to the wide availability of huge amounts of data and the imminent need for turning such data into useful information and knowledge (Han & Kamber, 2006). Since the problem of mining frequent patterns first introduced in Agrawal, Imieliinski, and Swami (1993), 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 (Agrawal & Srikant, 1994; Brin, Motwani, & Silverstein, 1997), cluster analysis (Agrawal, Gehrke, Gunopulos, & Raghavan, 1998; Beil, Ester, & Xu, 2002), classification (Liu, Hsu, & Ma, 1998; Wang & Karypis, 2005), and many other important data mining tasks (Li & Deng, 2010).

Although there are plenty of studies on pattern mining, such as in Han, Pei, and Yin (2000), Zaki and Gouda (2003) and Tzung-Pei et al. (2009), many new pattern mining problems have arisen, such as high-utility pattern mining (Hu & Mojsilovic, 2007), Mining of Discriminative and Essential Frequent Patterns (Fan et al., 2008), Approximate frequent pattern mining (Gupta, Fang, Field, Steinbach, & Kumar, 2008), concise Representation of Frequent itemsets (Jin, Xiang, & Liu, 2009; Poernomo & Gopalakrishnan, 2009), proportional fault-tolerant frequent itemsets mining (Poernomo & Gopalakrishnan, 2009), frequent pattern mining in uncertain Data (Aggarwal, Li, Wang, & Wang, 2009; Bernecker, Kriegel, Renz, Verhein, & Zuefl, 2009; Zhang, Li, & Yi, 2008), erasable itemsets mining (Deng, Fang, Wang, & Xu, 2009), 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 products 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. Deng et al. (2009) first introduced the problem of erasable itemsets mining and proposed an algorithm called META to deal with this problem.

Although META 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 is poor because it scans
database repeatedly. The second weakness is that META 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 structure called NC_set, which keeps track of the complete information used for mining erasable itemsets. Second, we propose an NC_set-based algorithm called MERIT for mining erasable itemsets efficiently. Note that the NC_set is similar to a data structure, Node-list (Deng & Wang, 2010). However, Node-lists are designed to facilitate the mining of frequent patterns.

Efficiency of our mining method is achieved with the following factors. First, a large product database is compressed into NC_sets, which are highly condensed and much smaller, without losing any information for mining erasable itemsets. Second, the gain of an itemset can be computed efficiently via combination operations on product NC_sets. Third, MERIT can directly find erasable itemsets without generating candidate itemsets in some cases. For evaluating this algorithm, we have conducted extensive experiments on a lot of synthetic product databases. Our performance study shows that MERIT is efficient and is on average about two orders of magnitude faster than META, which is the first algorithm for dealing with the problem of erasable itemsets mining. Our experimental results also show that the size of NC_sets is less than one-tenth of that of the original database.

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

2. Statement of problem

In this section, related concepts are described and a formal description of erasable itemset is given.

Let \( I = \{1, 2, \ldots, n\} \) 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 manufacture (or factory) reaps (or obtains) by selling all \( P_i \)-type products.

Table 1 shows an example product database. The database has eleven types of product. \( \{a, b, c, d, e, f, g, h\} \) is the set of universal items that comprise the eleven types of products. Let’s take \( P_3 \) as an example. We know that we need two different kind of item, which are \( a \) and \( c \), to manufacture \( P_3 \)-type products. It would profit 1000 million dollars by selling all \( P_3 \)-type products.

<table>
<thead>
<tr>
<th>Product</th>
<th>PID</th>
<th>Items</th>
<th>Val (Million $)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_1 )</td>
<td>1</td>
<td>a, b, c</td>
<td>2100</td>
</tr>
<tr>
<td>( P_2 )</td>
<td>2</td>
<td>a, b</td>
<td>1000</td>
</tr>
<tr>
<td>( P_3 )</td>
<td>3</td>
<td>a, c</td>
<td>1000</td>
</tr>
<tr>
<td>( P_4 )</td>
<td>4</td>
<td>d, b, c</td>
<td>150</td>
</tr>
<tr>
<td>( P_5 )</td>
<td>5</td>
<td>d</td>
<td>50</td>
</tr>
<tr>
<td>( P_6 )</td>
<td>6</td>
<td>d, c</td>
<td>100</td>
</tr>
<tr>
<td>( P_7 )</td>
<td>7</td>
<td>d, e, f, g, c</td>
<td>200</td>
</tr>
<tr>
<td>( P_8 )</td>
<td>8</td>
<td>d, e, f, h</td>
<td>100</td>
</tr>
<tr>
<td>( P_9 )</td>
<td>9</td>
<td>e, f</td>
<td>50</td>
</tr>
<tr>
<td>( P_{10} )</td>
<td>10</td>
<td>e, h, b</td>
<td>150</td>
</tr>
<tr>
<td>( P_{11} )</td>
<td>11</td>
<td>e, c</td>
<td>100</td>
</tr>
</tbody>
</table>

### Definition 1
Let \( A( \subseteq I) \) be an itemset (a set of items), the gain of \( A \) is defined as:

\[
Gain(A) = \sum_{(P_k \in A \text{ and } \text{Items} = P_k)} P_k \cdot \text{Val}.
\]

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 \( P = \{a = c\} \) be an itemset. From Table 1, we know that the products, who’s Items contain \( a \) or \( c \), are \( P_1, P_2, P_5, P_6, P_7, P_8, P_9, P_{10} \), and \( P_{11} \). Therefore, the gain of \( P \) is the sum of \( P_1 \cdot \text{Val}, P_2 \cdot \text{Val}, P_4 \cdot \text{Val}, P_6 \cdot \text{Val}, P_7 \cdot \text{Val}, \) and \( P_{11} \cdot \text{Val} \). That is, \( Gain(P) \) is 3650 million dollars.

### Definition 2
Given a predefined threshold \( \xi \) and a product database \( DB \), an itemset \( A \) is erasable if

\[
Gain(A) \leq \left( \sum_{P_k \in DB} P_k \cdot Val \right) \times \xi.
\]

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

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

Erasable itemsets are those items without which the profit of remain products is not less than \( 1 - \xi \) of that of initial products. For example, let \( \xi \) be 16%. Itemset \( d \) is an erasable. If a manufacture does not purchase \( d \) as raw materials because of economic crisis, the manufacture can not manufacture products that are type of \( P_1, P_2, P_3, P_4, P_5, \) or \( P_6 \). However, the lost profit is the \( \text{Gain (d)} \) (600 million dollars), which is no more than 16% of the original profit (5000 million dollars). Erasable itemsets is especially useful for managers to decide how to purchase raw materials and plan the process of manufacturing products in the case of economic crisis.

For the sake of discussion, an itemset with length of \( k \) is called a \( k \)-itemset in this paper. The length of an itemset is the number of items included in the itemset. Similarly, an erasable itemsets with length of \( k \) is called an erasable \( k \)-itemset. In this paper, an \( 1 \)-erasable itemset is also called an erasable item.

3. NC_set: definition, construction, and properties

In this section, we will describe the NC_set structure and some properties. Before the introduction of the NC_set, we first describe the WPPC-tree, which is the basic of the NC_set.

3.1. WPPC-tree

We define a WPPC-tree as follows.

### Definition 3
PPC-tree is a tree structure as follows:

(1) It consists of one root labeled as “null”, a set of item prefix subtrees as the children of the root.

(2) Each node in the item prefix subtree consists of five fields: item-name, weight, childTreeNode-list, pre-order, and post-order. item-name registers which frequent item this node represents. weight registers the sum of profits of all products presented by the portion of the path reaching this node. childTreeNode-list registers all children of the node. pre-order is the preorder rank of the node. post-order is the postorder rank of the node.
Based on Definition 1, we have the following WPPC-tree construction algorithm.

**Algorithm 1: WPPC-tree construction**

**Input:** A product database DB and a minimum support threshold \( \xi \)

**Output:** WPPC-tree, \( E_1 \) (the set of erasable 1-itemsets)

**Method:** Construct-WPPC-tree(DB, \( \xi \))

//Generate erasable 1-itemsets

(1) Scan DB once to find the set of erasable 1-itemset \( E_1 \) and their gains. Sort \( E_1 \) in frequency descending order as \( I_g \), which is the list of ordered frequent items. Note that the frequency of an item is the number of products that contain the item.

//Construct the WPPC-tree

(2) Create the root of a WPPC-tree, \( Tr \), and label it as "null". Scan DB again. For each product \( P \) in DB, arrange its erasable items into the order of \( I_g \). Without loss of generality, we still denote the arranged set of erasable items as \( P \). In addition, we denote the profit of \( P \) (that is \( P \cdot Val \)) by \( Val_p \). Call insert_tree(\( P, Tr \)) to insert it into the PPC-tree

//Generate the Pre-Post code of each node

(3) Scan PPC-tree to generate the pre-order and the post-order of each node

Function insert_tree(\( P, Tr \)) {
    While (\( P \) is not null) do {
        Let \( P[1] \) be the first element of \( P \) and \( P = P[1] \) means the remaining list after deleting \( P[1] \) from \( P \)
        If \( Tr \) has a child \( N \) such that \( N \cdot \text{item-name} = P[1] \), then increment \( N \cdot \text{weight} \) by \( Val_p \); else create a new node \( N \), and let its weight be \( Val_p \). Its parent link be linked to \( Tr \), and call insert_tree(\( P - P[1], N \))
    }
}

For better understanding of the concept and the construction algorithm of WPPC-tree, let's examine the following example.

**Example 1.** Let the transaction database, DB, as shown in Table 1 and \( \xi = 16\% \).

The WPPC-tree storing the DB is shown in Fig. 1. It should be noticed that, in fact, the PPC-tree is constructed via the product database showed by Table 2 according to algorithm 1. In the light of Definition 1, the gains of item \( a, b, \) and \( c \) are 4100, 3250, and 3500 million dollars. Therefore, they are inerasable items because their gains are larger than 800 million dollars, the 16 percents of whole profit (5000 million dollars). In the same way, item \( d, e, f, g, \) and \( h \) are erasable items. For the product database showed by Table 2, all inerasable items are eliminated and erasable items are listed in frequency-descending order. For mining erasable itemsets, the database showed by Table 2 is equivalent to the DB because they contain the same erasable items for each low (product). Note that \( P_1, P_2, \) and \( P_3 \) do not need to be considered for finding erasable itemsets because they do not contain any erasable itemsets at all.

After obtaining the pre-order and the post-order of each node by traversing the WPPC-tree, we get the result showed by Fig. 1. In this figure, the node with (2,3) means that its pre-order is 2, post-order is 3, and its item-name is \( e \), weight is 300.

3.2. NC_sets: definition

Before describing the definition of the NC_set, we first define an elementary concept, WPP-code, which is the only component that composes NC_set.

**Definition 4** (WPP-code). For each node \( N \) in the WPPC-tree, we call \( (N \cdot \text{pre-order}, N \cdot \text{post-order}): N \cdot \text{weight} \) as the WPP-code of \( N \).

In fact, constructing the WPPC-tree is to generate the WPP-codes of erasable items. The WPP-codes can effectively reflect the structure of the WPPC-tree, which is described by the following lemma (Grust, 2002).

**Lemma 1.** Given any two different nodes \( N_1 \) and \( N_2 \) in a WPPC-tree, \( N_1 \) is an ancestor of \( N_2 \) if and only if \( N_1 \cdot \text{pre-order} = N_2 \cdot \text{pre-order} \) and \( N_1 \cdot \text{post-order} \neq N_2 \cdot \text{post-order} \).

It is decided by the construction of pre-order and post-order. When \( N_1 \) is an ancestor of \( N_2 \), \( N_2 \) must be traversed earlier than \( N_1 \) during the preorder traversal and be traversed later than \( N_2 \) during the postorder traversal. On the other side, if \( N_1 \cdot \text{pre-order} = N_2 \cdot \text{pre-order} \) and \( N_1 \cdot \text{post-order} \neq N_2 \cdot \text{post-order} \), \( N_1 \) is the node that is traversed earlier than \( N_2 \) during the preorder traversal and later during the postorder traversal.

By using Lemma 1, it is easy to find the ancestor-descendant relationship of any two nodes just based on the pre-order and post-order. Lemma 1 also shows that nodes and their WPP-codes are 1–1 mapping. That is, a node uniquely determines a WPP-code and a WPP-code also uniquely determines a node. In fact, a node and its PP-code are equivalent. Therefore, we also denote a node by its PP-code in this paper.

Based Lemma 1, we have the following lemma.

**Lemma 2.** The ancestor-descendant relationship of WPP-codes

Let \( X_1 ((x_1, y_1): z_1) \) and \( X_2 ((x_2, y_2): z_2) \) be two WPP-codes, \( X_1 \) is an ancestor of \( X_2 \) if and only if \( x_1 < x_2 \) and \( y_1 > y_2 \).

For example, \( (1,4): 600 \) is an ancestor of \( (3,2): 300 \) because of \( 1 < 3 \) and \( 4 > 2 \).
Definition 5 (the NC_set of an erasable item). Given a WPPC-tree, the NC_set of an erasable item is a sequence of all the WPP-codes of nodes registering the item in the WPPC-tree. The WPP-codes are arranged according to the pre-order descending order.

According to Definition 5, the NC_set of an erasable item is denoted by \((\{x_1, y_1\}, \{x_2, y_2\}, \ldots, \{x_n, y_n\}\)\), where \(x_k < x_{k+1} < \cdots < x_n\). For example, the Node-list of \(d\) includes one node. Its pre-order is 1, its post-order is 4, and its weight is 600. Fig. 2 shows the NC_set of all erasable items in Example 1.

We denote the set of erasable items by \(I_d\), in which erasable items are sorted in their frequency descending. For the definition of frequency of an erasable item, please refer to step (1) of Algorithm 1.

For example, \(I_d\) of example 1 is \(\{d, e, f, h, g\}\). \(d\) is the first element because its frequency is 5, which is biggest. Note that if two or more items have the same frequency, we specify an order, such as lexicographical order, for them.

Based on \(I_d\), we define \(\succ\) relation of two items as follows.

Definition 6 (\(\succ\) Relation). For any two erasable items \(i_1\) and \(i_2\), we call \(i_1 \succ i_2\) if and only if \(i_1\) is ahead of \(i_2\) in \(I_d\).

For the sake of discussion, any itemset \(O\) in this paper is denoted by \(i_1i_2\ldots i_k\), where \(i_1 \succ i_2 \succ \cdots \succ i_k\). For example, we represent \(O\) by \(df\) if \(O\) only contains \(d\) and \(f\).

Next, let’s extend Definition 5 to the concept of the NC_set of a k-itemset (\(k > 2\)).

Definition 7 (The NC_set of a k-itemset). Let \(O = i_1i_2\ldots i_k\) be an itemset (\(k > 2\)), and the NC_set of \(O_1 = i_1i_2\ldots i_k\) is \(\{(x_{11}, y_{11}), (x_{12}, y_{12}), \ldots, (x_{1m}, y_{1m})\}\), the NC_set of \(O_2 = i_1i_2\ldots i_k\ldots i_{k-1}\ldots i_1\ldots i_k\) is \(\{(x_{21}, y_{21}), (x_{22}, y_{22}), \ldots, (x_{2n}, y_{2n})\}\). The NC_set of \(O\), \(\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}\), is a sequence of WPP-codes by combining the NC_sets of \(O_1\) and \(O_2\), which meets the following conditions:

1. The elements in NC_set of \(O\) are arranged in pre-order ascending order. That is, \(x_i < x_j\) for \(k < j\).
2. For any \(\{(x_{1k}, y_{1k})\}\) \((1 \leq k \leq m)\) in the NC_set of \(O_1\), it must be an element of the NC_set of \(O\).
3. Let \(\{(x_{2k}, y_{2k})\}\) be an element of the Node-list of \(O_2\) \((1 \leq k \leq n)\), if no element in the NC_set of \(O_1\) is its ancestor, it must be an element of the NC_set of \(O\).

Note that the definition of the NC_set of a k-itemset is recursive. For computing the NC_set of a k-itemset we must obtain the NC_set of relevant \((k - 1)\)-itemset.

For example, in Fig. 2, the NC_set of \(d\) is \(\{(1,4):600\}\) and the NC_set of \(e\) is \(\{(2,3):300, (6,7):300\}\). According to Definition 7, \(\{(1,4):600\}\) must be in the NC_set of \(de\). According to Definition 5, \(\{(1,4):600\}\) is an ancestor of \(\{(2,3):300\}\) because of \(1 < 2\) and \(4 > 3\). Therefore \(\{(2,3):300\}\) is not an element of the NC_set of \(de\). However, \(\{(1,4):600\}\) is not an ancestor of \(\{(6,7):300\}\) because of \(4 < 7\). So, \(\{(6,7):300\}\) must be in the NC_set of \(de\) in the light of Definition 7. Finally, we know that the NC_set of \(de\) is \(\{(1,4):600, (6,7):300\}\). Fig. 3 depicts the processing.

One advantage of NC_set format is that we can very convenient to obtain the gain of itemsets. For example, we know the gain of \(d\) is 600 million dollars. It is interesting that the weight of \(\{(1,4):600\}\), which is the only element of the NC_set of \(d\), is also 600. The gain of \(de\) is 900 million dollars. Surprisingly, the sum of the weights of \(\{(1,4):600\}\) and \(\{(6,7):300\}\) is also 900 (600 + 300). From Fig. 3, we know the NC_set of \(de\) is \(\{(1,4):600, (6,7):300\}\). The two examples mean that we can obtain the gain of an itemset just by adding up weights of all WPP-codes in the NC_set of the itemset. In fact, this conclusion is correct. The next section will proves it.

3.3. NC_sets:

Property 1. Let \(i\) be an erasable 1-itemset and its NC_set is \(\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}\). The gain of \(i\) can be computed as follows.

\[\text{Gain}(i) = \sum_{j=1}^{n} z_j.\]  

Proof. Let \(DB_i\) be the set of products that contain \(i\) and register \(i\) in node \((x_j, y_j)\). According to the construction of WPPC-tree, we have

(a) For \(j \neq k\), \(DB_i \cap DB_k = \emptyset\);
(b) For any product \(P\) containing \(i\), there must exists a \(DB_j\) \((1 \leq s \leq n)\) to which \(P\) belongs, that is \(P \in DB_j\).

Let \(DB\) be the set of all products containing \(i\). According to (a) and (b), we have

\[DB = \bigcup_{j=1}^{n} DB_j.\]  

So, according to Definition 1, we have

\[\text{Gain}(i) = \sum_{\{P_j\} \in \{P_i\}^{\subseteq} \text{Val}.\]  

(5)

Then, we can reformulate (5) as follows:

\[\text{Gain}(i) = \sum_{\{P_j\} \in \{P_i\}^{\subseteq} \text{Val}.\sum_{\{P_k\} \in \{P_j\}^{\subseteq} \text{Val}.\sum_{j=1}^{n} P_k \cdot \text{Val} = \sum_{j=1}^{n} \sum_{P_j} \text{Val}.\]  

(6)

So, we prove Property 1. □

In fact, for any 1-itemset, Property 1 is still valid because our proof does not care whether 1-itemset is erasable or not.
Next, we extend Property 1 to the following conclusion.

**Property 2.** Let \( O = i_1i_2...i_{k-2}...i_k \) be a \( k \)-itemset and its NC_set is \( \{(x_1,y_1): z_1\}, \{(x_2,y_2): z_2\}, ..., \{(x_o,y_o): z_o\} \). The gain of \( p \) can be computed as follows.

\[
Gain(O) = \sum_{j=1}^{l} z_j.
\]

(6)

**Proof.** We will prove this property by mathematical induction.

If \( P \) be 1-itemset, we know Eq. (6) is valid according to Property 1.

Let’s assume that Eq. (6) is valid for any \((k-1)\)-items. Then, we prove that for any \( k \)-itemset Eq. (6) is still valid.

According to Definition 1, we have

\[
Gain(O) = \sum_{\{P_j|O \cap P_j \neq \phi\}} P_j \cdot Val.
\]

(7)

Let \( O_1 = i_1i_2...i_{k-2}...i_k \), we have

\[
\{P_j|O \cap P_j \neq \phi\} = \{P_j|O_1 \cap P_j \neq \phi\} \setminus \{P_j|O_1 \cap P_j = \phi\}
\]

\[
= \{P_j|O_1 \cap P_j \neq \phi\} \land \{i_k \in P_j \cdot Items\}.
\]

(8)

According to the construction of WPPC-tree, only products that contain \( i_k \), but do not contain \( i_1, i_2, ..., i_{k-2} \), and \( i_{k-1} \) register \( i_k \) and its profit in a node corresponding to some element in NC(\( i_k \)). The set of these products is SS2 according to the definition of SS2. So, we have

\[
\sum_{SS2} P_j \cdot Val = \sum_{\{(x,y) \in NC(i_k) \cup NC(i_k): z1\}} z1.
\]

(15)

Combining Eqs. (13), (14), (15) and (9), we have

\[
\sum_{j=1}^{k} z_j = \sum_{\{(x,y) \in NC(i_k-i_k)\}} z1 = \sum_{\{(x,y) \in NC(i_k-i_k)\}} z1 \cdot Gain(O).
\]

(16)

So, we prove Property 2. \( \Box \)

4. Mining erasable itemsets using NC_sets

Deng et al. (2009) indicate that erasable itemsets has a characteristic called anti-monotone as shown by the following property.

**Property 3 (Anti-monotone).** If itemset \( X \) is inerasable, and \( Y \) is a superset of \( X \) \((X \subseteq Y)\), \( Y \) must also be inerasable.

Based on Property 3, our algorithm, MERIT, employs Apriori-like approach to mine erasable itemsets. First, we generate the NC_sets of candidate \((k)\)-itemsets by combining the NC_sets of erasable \((k-1)\)-itemsets. Second, for any candidate \((k-1)\)-itemsets \( O \), we obtain the gain of \( O \) by summing weight values of all WPPC-codes in its NC_set. According to \( O \)’ gain, we can judge whether \( O \) is erasable or not. By repeating the above procedure, all erasable itemsets will be found.

Obviously, the efficiency of combining two NC_sets is vital to the efficiency of MERIT. Therefore, we first discuss how to devise efficient method for combining NC_sets. Then, we describe the details of MERIT.

4.1. An efficient method for combining NC_sets

Let \( O_1 = i_1i_2...i_{k-2}...i_k \) and \( O_2 = i_1i_2...i_{k-2}...i_k \) be two \((k-1)\)-itemsets. The NC_set of \( O_1 \) is \( \{(x_1,y_1): z1\}, \{(x_2,y_2): z1\}, ..., \{(x_{k-1},y_{k-1}): z1\} \). The NC_set of \( O_2 \) is \( \{(x_{k-1},y_{k-1}): z1\}, \{(x_{k-2},y_{k-2}): z1\}, ..., \{(x_2,y_2): z1\}, \{(x_1,y_1): z1\}, \{(x_1,y_1): z1\} \). For generating the NC_set of \( O = i_1...i_{k-2}...i_k \), a Naïve method is to check each WPPC-code of the Node-list of \( P_2 \) with each WPPC-codes of the Node-list of \( P_1 \) to decide whether they satisfy the ancestor–descendant relationship. It is obvious that the time complexity of the Naïve method is \( O(mn) \). This time complexity is unsatisfying. After some careful
analysis, we find a linear-time-complexity method. Before presenting our method, let's first introduce some lemmas.

**Lemma 3.** Let \( i \) be an erasable 1-itemset, and \((x_1, y_1); z_1\) and \((x_2, y_2); z_2\) be two elements in its NC_set. We have that neither \((x_1, y_1); z_1\) is an ancestor of \((x_2, y_2); z_2\) nor \((x_2, y_2); z_2\) is an ancestor of \((x_1, y_1); z_1\). That is, they have no ancestor–descendant relationship.

**Proof.** According to the construction of WPPC-tree, nodes registering the same item can not have ancestor–descendant relationship. Item \( i \) is registering by the Node corresponding to \((x_1, y_1); z_1\) and the node corresponding to \((x_1, y_1); z_1\). Therefore, \((x_1, y_1); z_1\) and \((x_2, y_2); z_2\) cannot have ancestor–descendant relationship. □

Lemma 3 can be extended to a more general form as follows.

**Lemma 4.** Let \( O = i_1 i_2 \ldots i_k \) be a k-itemset, and \((x_1, y_1); z_1\) and \((x_2, y_2); z_2\) be two elements in its NC_set. We have that neither \((x_1, y_1); z_1\) is an ancestor of \((x_2, y_2); z_2\) nor \((x_2, y_2); z_2\) is an ancestor of \((x_1, y_1); z_1\).

**Proof.** First, we know any two elements in the NC_set of an erasable 1-itemset can not have the ancestor–descendant relationship according to Lemma 3. Second, let \( i_1 \) and \( i_2 \) be two erasable 1-itemsets and \( i_1 > i_2 \). Assume \( E_1 \) is an element in the NC_set of \( i_1 \) and \( E_2 \) is an element in the NC_set of \( i_2 \). According to the construction of the WPPC-tree, \( E_2 \) can not be an ancestor of \( E_1 \). Finally, combining the above conclusions and Definition 7, Lemma 4 is established. □

**Lemma 5.** Let \( O = i_1 i_2 \ldots i_k \) be a k-itemset and the NC_set of \( O \) is \( \{ (x_1, y_1); z_1, (x_2, y_2); z_2, \ldots, (x_m, y_m); z_m \} \). We have \( x_1 < x_2 < \ldots < x_m \) and \( y_1 < y_2 < \ldots < y_m \).

**Proof.** According to Definition 7, we have \( x_1 < x_2 < \ldots < x_m \). We know \( x_i < x_j \) because of \( s < t \). So, we have \( x_i < y_i \). According to Lemma 2, \((x_i, y_i); z_i\) must be an ancestor of \((x_1, y_1); z_1\). However, Lemma 4 indicates that any two elements in a NC_set can not have the ancestor–descendant relationship. Therefore, our assumption is wrong. That is, Lemma 5 is established. □

**Lemma 6.** Let \( O_1 = i_1 i_2 \ldots i_k \) and \( O_2 = i_1 i_2 \ldots i_k \) be two \((k-1)\)-itemsets with \( i_k > i_1 \). \( O_1 \)'s NC_set is denoted by \( \{ (x_1, y_1); z_1, (x_2, y_2); z_2, \ldots, (x_m, y_m); z_m \} \) and \( O_2 \)'s NC_set is denoted by \( \{ (x_1, y_1); z_1, (x_2, y_2); z_2, \ldots, (x_m, y_m); z_m \} \). If \((x_1, y_1); z_1\) is an ancestor of \((x_2, y_2); z_2\), then for any \((x_j, y_j); z_j\) with \( j \neq s \) can not be an ancestor of \((x_2, y_2); z_2\).

**Proof.** Assume there exist \((x_j, y_j); z_j\) \((j \neq s)\) that is an ancestor of \((x_2, y_2); z_2\). According to the construction of the WPPC-tree, the node corresponding to \((x_2, y_2); z_2\) and \((x_1, y_1); z_1\) must have ancestor–descendant relationship. That is, \((x_1, y_1); z_1\) and \((x_1, y_1); z_1\) have ancestor–descendant relationship. This conflicts with Lemma 4. □

Now, let’s see how to generate \( O \)'s NC_set. According to Definition 7, every element of \( O_1 \)'s NC_set must be an element of \( O \)'s NC_set. Therefore, it’s vital for us to know which elements in \( O_2 \)'s NC_set should also belong to \( O \)'s NC_set. Our method selects elements from \( \{ (x_1, y_1); z_1, (x_2, y_2); z_2, \ldots, (x_2, y_2); z_2 \} \) according to the order from left to right. Then, we check the ancestor–descendant relationship of the element and elements in \( \{ (x_1, y_1); z_1, (x_2, y_2); z_2, \ldots, (x_m, y_m); z_m \} \) consecutively. Let \((x_1, y_1); z_1\) and \((x_2, y_2); z_2\) be the current WPP-codes to be proceeded. That is, we have find no \((x_1, y_1); z_1\) \((s < i)\) is an ancestor of \((x_2, y_2); z_2\) and are checking whether \((x_1, y_1); z_1\) is an ancestor of \((x_2, y_2); z_2\).

With different values of \( x_1, y_1, x_2, y_2 \), there would be three cases: \( x_1 > x_2, (x_1 < x_2) \) \((y_1 < y_2)\), and \((x_1 < x_2) \) \((y_1 > y_2)\). Note that \( x_1 \) and \( x_2 \) are the pre-order of different nodes in the WPPC_tree. Therefore, \( x_1 \) can not be equal to \( x_2 \). In the same way, \( y_1 \) can not be equal to \( y_2 \). According to different cases, our process is as follows:

1. According to Definition 7, \((x_1, y_1); z_1\) should be a element of \( O \)'s NC_set. Therefore, we insert \((x_1, y_1); z_1\) into \( O \)'s NC_set.

2. The case of \( x_1 > x_2 \). According to Lemma 2, \((x_1, y_1); z_1\) can not be an ancestor of \((x_2, y_2); z_2\). According to Lemma 5, we know \( x_1 < x_2 \). Therefore, \((x_1, y_1); z_1\) \((s < i)\) is an ancestor of \((x_2, y_2); z_2\) with these elements. Therefore, we should insert \((x_2, y_2); z_2\) into \( O \)'s NC_set and go on comparing \((x_2, y_2); z_2\) with \((x_1, y_1); z_1\).

3. The case of \( x_1 < x_2 \). According to Lemma 2, \((x_1, y_1); z_1\) can not be an ancestor of \((x_2, y_2); z_2\). However, in the light of Lemma 5, we know \( y_1 < y_2 \). So, some \( y_1 \) \((s < i)\) can not be bigger than \( y_2 \). Therefore, we should go on comparing \((x_2, y_2); z_2\) with \((x_1, y_1); z_1\).

4. The case of \( x_1 < x_2 \) \((y_1 > y_2)\). According to Lemma 2, \((x_1, y_1); z_1\) is an ancestor of \((x_2, y_2); z_2\). Lemma 6 indicates that all other elements in \( O \)'s NC_set can not be an ancestor of \((x_2, y_2); z_2\). Therefore, we should go on comparing \((x_2, y_2); z_2\) with \((x_1, y_1); z_1\) \((s < i)\). According to the processing, \( y_1 \) is the first one that is bigger than \( y_2 \). This means \( y_1 < y_2 \). Therefore, we have \( y_1 < y_2 \). According to Lemma 2, \((x_1, y_1); z_1\) \((s < i)\) can not be an ancestor of \((x_2, y_2); z_2\).

Based on the above analysis, we have the following algorithm.

**Algorithm 2: NC_Combination**

**Input:** \( NL_1 = \{ (x_1, y_1); z_1, (x_2, y_2); z_2, \ldots, (x_m, y_m); z_m \} \) and \( NL_2 = \{ (x_1, y_1); z_1, (x_2, y_2); z_2, \ldots, (x_m, y_m); z_m \} \) which are the N-lists of \( O_1 \) and \( O_2 \) respectively.

**Output:** \( NL_3 \), the N-list of \( O = i_1 i_2 \ldots i_k \)

**Procedure NC_Combination:**

```plaintext
NL ← Ø;
c ← 1;
for j = 1 to m do {
  if (\( x_j > x_c \)) \((c \leq n)\) do {
    insert \((x_c, y_c); z_c\) into NL;
    c++
  }
  insert \((x_j, y_j); z_j\) into NL;
  if (\( y_j > y_c \)) \((c \leq n)\) do {
    c++
  }
}
```

```
if (\( c \leq n \)) then {
  For j = c to n do {
    quad insert \((x_c, y_c); z_c\) into NL
  }
}
```

In fact, our method makes use of the characteristic that WPP-codes in a NC-set are ordinal. Obviously, the worst running time of our method is $m + n$.

4.2. MERIT algorithm

Before describing MERIT algorithm, let’s see a very important property which is useful for efficiently reducing candidate generation in the mining procedure.

Let $O_1 = i_1i_2...i_k...i_{k-2}...i_1$, $O_2 = i_1i_2...i_{k-2}...i_1$, ..., and $O_6 = i_{12}...i_{k-2}...i_1$, $l_{12} > l_{12} > ... > l_6$, be $(k - 1)$-itemsets, and $O_1$, NC-sets denoted by $NC_1$. Moreover, we denote $i_1i_2...i_{k-2}...i_1$ $(2 \leq j \leq s)$ by $O_j$ and its NC-set by $NC_1$.

We have the following lemma.

**Lemma 7.** If for any $j$ $(2 \leq j \leq s)$ NC$_j$ is equal to NC$_1$, then for any $i_1i_2...i_{k-2}...i_1$, $l_{12} < ... < l_{12} < l_{12}$ $(2 \leq v_1 < v_2 < ... < v_s \leq s)$, its NC-set is also equal to NC$_1$.

**Proof.** For $x = 1$. The NC-set of $i_1i_2...i_{k-2}...i_1$ $(2 \leq v_1 \leq s)$ is NC$_1$. According to the lemma’s prerequisite, NC$_1$ is equal to NC$_1$.

For $x = 2$. The NC-set of $i_1i_2...i_{k-2}...i_1$ $(2 \leq v_1 < v_2 \leq s)$ is generated by combining the NC-set of $i_1i_2...i_{k-2}...i_1$ and the NC-set of $i_1i_2...i_{k-2}...i_1$. However, we know their NC-sets are equal to NC$_1$. Therefore, according to Definition 7, The NC-set of $i_1i_2...i_{k-2}...i_1$ is equal to NC$_1$.

In the same way, we can proof that Lemma 7 is valid for $x \geq 3$ by repeating the similar processing procedures used for $x = 2$.

Based on the discussion above, Lemma 7 is established. □

Base on Lemma 7, we have the following Corollary 1.

**Corollary 1.** If $O_1(i_1i_2...i_{k-2}...i_1)$ is erasable, $i_1i_2...i_{k-2}...i_1$ is erasable $(2 \leq v_1 < v_2 < ... < v_s \leq s)$ is also erasable.

Lemma 7 and Corollary 1 indicate that in some cases we can directly find lots of erasable itemsets without any combining operation. By employing these conclusions, we can greatly improve the efficiency of our algorithm.

For better understanding Lemma 7 and Corollary 1, let’s examine the example shown by Fig. 4. Note that, limited by space, we use rectangles to represent WPP-codes instead of () in Fig. 4. The NC-sets of $e$, $eh$, and $eg$ are $((2, 3): 300), ((6, 7): 300)$, which is the NC-set of $e$. Repeatedly employing Definition 7, we can obtain the NC-sets of $efh$, $egf$, $ehg$, and $efgh$. They are all equal to $((2, 3): 300), ((6, 7): 300)$.

However, based on Lemma 7, we directly know the NC-sets of $efh$, $egf$, $ehg$, and $efgh$. In addition, if $e$ is an erasable item, we can confirm that $efh$, $egf$, $ehg$, and $efgh$ must be erasable itemsets without any combining operation.

Based on all discussions above, we have the following algorithm, MERIT, for fast mining erasable itemsets.

![Fig. 4. An example for illustrating Lemma 7.](image-url)

**Algorithm 3: MERIT**

**Input:** the product database, $DB$, and the minimum support threshold, $\xi$.

**Output:** $E$, the set of all erasable itemsets, and $NC$, the set of the NC-sets of all erasable itemsets in $E$.

1. Call **Construct-WPPC-tree** $(DB, \xi)$ to generate the tree, **WPPC**, and the set of erasable 1-itemsets, $E_1$.
2. $E = E_1$.
3. Scan **WPPC** by preorder traversal to generate $NC_1$, the set of NC-sets of all erasable 1-itemsets.
4. $NC = NC_1$.
5. If $E_1.size > 1$, then call **mining_E** $(E_1, NC_1)$.
6. Retrun $E$ and $NC$.

Procedure **mining_E**(EC, NC)

$\forall E$ is the set of equivalence classes of erasable itemsets of which only the last item is different, and $NC$ is the set of NC-sets of all equivalence classes of erasable itemsets in $E$. For example, the element $[i_1i_2]$ of $E$ represents the set of erasable itemsets, which must include $i_1$ and $i_2$ and may have other items but all have the same NC-set. Let $NC_1 = \{i_1, i_2\}$ and $NC_2 = \{i_1, i_2\}$, and we call “$i_1i_2$” the representation itemsets of the equivalence classes $\{\}

1. for $k = 1$ to $EC.size$ do
2. $EC.next = 0$;
3. $NC.next = 0$;
4. for $j = (k + 1)$ to $EC.size$ do
5. let $R_k$ be the representation itemsets of $EC_k$, and let $R_k$ be the representation itemsets of $EC_j$;
6. $Cd = R_k \cup R_j$;
7. if any subset of $Cd$ with length $(Cd.size - 1)$ is erasable, then
8. $Cd$ is the set of erasable itemsets, $NC$ is the set of erasable itemsets, and $NC_k$ is the set of NC-sets of all erasable itemsets in $E$.
9. for $Cd$ gain $\text{DB}.profit \times \xi$ then
10. **EC.next** = **EC.next** $\cup \{[Cd]\};$
11. **NC.next** = **NC.next** $\cup \{\text{NC_set}\}$
12. } end if
13. } end if
14. } end for
15. Scan **NC.next** to find all itemsets in **EC.next** whose NC-set is equal to the NC-set of $EC_k$, and use these itemsets to enlarge the equivalence class $EC_k$, and also remove them out of **EC.next** and remove the corresponding NC-sets out of **NC.next**.
16. Update equivalence classes of **EC.next** due to the change of $EC_k$.
17. if $EC.size > 1$, then
18. Call **mining_E**(EC$_{E.next}$, NC$_{E.next}$)
19. } end if
20. } end for
21. $EC = EC \cup EC$;
22. $NC = NC \cup NC$.}

Note that $EC_k$ and $EC_j$ represents the $k$th element and $j$th element in $E$, respectively. Let $EC_k$ be $i_1i_2...i_{k-2}...i_1$ $(i_1 > i_2 > ... > i_{k-2} > i_1)$ and $EC_j$ be $(i_1 > i_2 > ... > i_{k-2} > i_1)$. We define $EC_k \otimes EC_j$ as follows:

$$EC_k \otimes EC_j = \begin{cases} i_1i_2...i_{k-2}...i_1 & x_k > x_j; \\ i_1i_2...i_{k-2}...i_1 & x_k < x_j. \end{cases}$$  \hspace{1cm} (17)$$

Procedure **mining_E** $()$ adopts a strategy mixing depth-first search and breadth-first search to mine erasable itemsets. Step (7) employs Property 3 (anti-monotone) to prune unwanted candidate itemsets. Step (15) and step (16) employ Corollary 1 to directly find erasable itemsets without generating candidate itemsets.
5. Experimental evaluation and performance study

In this section, we present a performance comparison of MERIT with META algorithm for mining erasable itemsets. All the experiments were performed on a Dell PC with Intel Core2 Duo 2.66G and 2G Memory. The operating system was Microsoft Windows XP. All the programs were coded in MS/Visual C++. Note that runtime used here means the total execution time. That is, it is the period between input and output.

The experiments are conducted on synthetic databases. For obtaining these databases, we first generate them by IBM generator (Agrawal & Srikant, 1994). 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 product 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.

In the following subsections, we first report the compactness of NC_sets. Then, we report the experiments in which the runtime of MERIT was compared with META. Finally, we report the scalability study.

5.1. Compactness of NC_sets

In MERIT algorithm, all erasable items originated from erasable 1-items (or 1-itemsets). Therefore, the size of erasable 1-items is one of import factors that affect the efficiency of MERIT. To test the compactness of erasable 1-items, we have conducted many experiments on the six databases with different thresholds. Limited by space, we only show part of our experimental results here. Note that all our experimental results show the similar conclusion that NC_sets achieves good compactness.

Fig. 5 shows the comparison between the original size of the six databases and the size of corresponding erasable 1-items. Note that, for the sake of brevity, T15N50_25, T15U1_100, T20N50_25, T20U1_100, T30N50_25, and T30U1_100 are denoted by DB1, DB2, DB3, DB4, DB5, and DB6 respectively in Fig. 5. For DB1 and DB2, the threshold is 0.025. For DB3 and DB4, the threshold is 0.04. For DB5 and DB6, the threshold is 0.06. No matter which database is selected, NC_sets achieves good compactness. As seen from the result shown in Fig. 5, the size of NC_sets is always about an order of magnitude smaller than the size of the corresponding original database. In fact, the good compactness of NC_sets makes the search space smaller. This directly facilitates the mining process of MERIT algorithm.

5.2. The runtime

In Figs. 6–11, we compare MERIT against META for increasing values of threshold on different databases. The range of threshold is different. For T15N50_25 and T15U1_100, the range of threshold is from 0.01 to 0.07. For T20N50_25 and T20U1_100, the range of threshold is from 0.02 to 0.10. For T30N50_25 and T30U1_100, the range of threshold is from 0.03 to 0.15.

When the average product size becomes lower, the frequency of item assembled into a production gets lower. As a result, more erasable itemsets will be mined and more running time will be taken. That is why in the experiments we choose different range of the threshold for different databases. For databases with short product size, we set low thresholds, and vice versa.

From Fig. 6–11, we observe that both the running time of two algorithms increases when the threshold increases. We also find the following phenomena. For low threshold, the advantage of MERIT over META is not much notable. However, the difference between MERIT and META is more and more remarkable as the threshold increases. In each figure, MERIT is about an order of magnitude faster than META for the second minimum threshold. For the largest two thresholds, MERIT is over two orders of magnitude faster than META. That is, compared with META, the running time of MERIT algorithm increases at a slower rate. These six figures consistently show that MERIT algorithm always outperform META over different databases.

The reason that MERIT perform much better than META can be explained by three factors. The first one is that NC_sets is a good compressed structure without losing any information used for mining erasable itemsets and provide a natural way to prune irrelevant data automatically. The second one is that computing the gain of an itemset via NC_sets is much efficient than scanning original databases. The last one is that MERIT avoids generating candidate itemsets in some case by employing Corollary 1. It is very helpful to improve the efficiency of MERIT.

From our experimental results, we also find the probability distribution of profits has little to do with the runtime. This is well understood. Different probability distributions just assign different values (profits) to products. Mining algorithm use only a basic mathematic operator, plus, on these values. So, the change of values has little effect on the runtime.

5.3. Scalability

To test the scalability of MERIT and META, we conduct two groups of experiments. The first group is to test the scalability of MERIT and META against the size of databases. The second group is to test the scalability of MERIT and META against the number of items, that is, the size of the set of universal items.

Table 3
The Summary of databases used in our evaluation.

<table>
<thead>
<tr>
<th>Database</th>
<th># Product</th>
<th># Items</th>
<th>Probability distribution of profits</th>
</tr>
</thead>
<tbody>
<tr>
<td>T15N50_25</td>
<td>100,000</td>
<td>200</td>
<td>U(1,100) N(50,25)</td>
</tr>
<tr>
<td>T15U1_100</td>
<td>100,000</td>
<td>200</td>
<td>U(1,100) N(50,25)</td>
</tr>
<tr>
<td>T20N50_25</td>
<td>100,000</td>
<td>200</td>
<td>U(1,100) N(50,25)</td>
</tr>
<tr>
<td>T20U1_100</td>
<td>100,000</td>
<td>200</td>
<td>U(1,100) N(50,25)</td>
</tr>
<tr>
<td>T30N50_25</td>
<td>100,000</td>
<td>200</td>
<td>U(1,100) N(50,25)</td>
</tr>
<tr>
<td>T30U1_100</td>
<td>100,000</td>
<td>200</td>
<td>U(1,100) N(50,25)</td>
</tr>
</tbody>
</table>

Fig. 5. Size Comparison of NC_sets over six databases.

In the following subsections, we first report the compactness of NC_sets. Then, we report the experiments in which the runtime of MERIT was compared with META. Finally, we report the scalability study.

5.1. Compactness of NC_sets

In MERIT algorithm, all erasable items originated from erasable 1-items (or 1-itemsets). Therefore, the size of erasable 1-items is one of import factors that affect the efficiency of MERIT. To test the compactness of erasable 1-items, we have conducted many experiments on the six databases with different thresholds. Limited by space, we only show part of our experimental results here. Note that all our experimental results show the similar conclusion that NC_sets achieves good compactness.

Fig. 5 shows the comparison between the original size of the six databases and the size of corresponding erasable 1-items. Note that, for the sake of brevity, T15N50_25, T15U1_100, T20N50_25, T20U1_100, T30N50_25, and T30U1_100 are denoted by DB1, DB2, DB3, DB4, DB5, and DB6 respectively in Fig. 5. For DB1 and DB2, the threshold is 0.025. For DB3 and DB4, the threshold is 0.04. For DB5 and DB6, the threshold is 0.06. No matter which database is selected, NC_sets achieves good compactness. As seen from the result shown in Fig. 5, the size of NC_sets is always about an order of magnitude smaller than the size of the corresponding original database. In fact, the good compactness of NC_sets makes the search space smaller. This directly facilitates the mining process of MERIT algorithm.

5.2. The runtime

In Figs. 6–11, we compare MERIT against META for increasing values of threshold on different databases. The range of threshold is different. For T15N50_25 and T15U1_100, the range of threshold is from 0.01 to 0.07. For T20N50_25 and T20U1_100, the range of threshold is from 0.02 to 0.10. For T30N50_25 and T30U1_100, the range of threshold is from 0.03 to 0.15.

When the average product size becomes lower, the frequency of item assembled into a production gets lower. As a result, more erasable itemsets will be mined and more running time will be taken. That is why in the experiments we choose different range of the threshold for different databases. For databases with short product size, we set low thresholds, and vice versa.

From Fig. 6–11, we observe that both the running time of two algorithms increases when the threshold increases. We also find the following phenomena. For low threshold, the advantage of MERIT over META is not much notable. However, the difference between MERIT and META is more and more remarkable as the threshold increases. In each figure, MERIT is about an order of magnitude faster than META for the second minimum threshold. For the largest two thresholds, MERIT is over two orders of magnitude faster than META. That is, compared with META, the running time of MERIT algorithm increases at a slower rate. These six figures consistently show that MERIT algorithm always outperform META over different databases.

The reason that MERIT perform much better than META can be explained by three factors. The first one is that NC_sets is a good compressed structure without losing any information used for mining erasable itemsets and provide a natural way to prune irrelevant data automatically. The second one is that computing the gain of an itemset via NC_sets is much efficient than scanning original databases. The last one is that MERIT avoids generating candidate itemsets in some case by employing Corollary 1. It is very helpful to improve the efficiency of MERIT.

From our experimental results, we also find the probability distribution of profits has little to do with the runtime. This is well understood. Different probability distributions just assign different values (profits) to products. Mining algorithm use only a basic mathematic operator, plus, on these values. So, the change of values has little effect on the runtime.

5.3. Scalability

To test the scalability of MERIT and META, we conduct two groups of experiments. The first group is to test the scalability of MERIT and META against the size of databases. The second group is to test the scalability of MERIT and META against the number of items, that is, the size of the set of universal items.
5.3.1. Against the size of databases

In these experiments, we keep the item number, the average product size and the probability distribution of product profit constant and vary the database size from 20 k to 100 k. In all databases, the number of items, \( N_I \), is 200 and the probability distribution of product profit, \( PD \), is \( N(50,25) \). The average product length, \( APL \), of databases used in Figs. 12–14 are 15, 20, and 30 respectively. In Fig. 12, MERIT and META ran on five databases for threshold 0.04. In Fig. 13, MERIT and META ran on five databases for threshold 0.06.

Fig. 12. Scalability with fixing \( N_I = 200, APL = 15, PD = N(50,25) \), and Threshold = 0.04.

Fig. 13. Scalability with fixing \( N_I = 200, APL = 20, PD = N(50,25) \), and Threshold = 0.06.

Fig. 14. Scalability with fixing \( N_I = 200, APL = 30, PD = N(50,25) \), and Threshold = 0.09.
abases for threshold 0.06. MERIT and META ran on five databases for threshold 0.09 in Fig. 14. Figs. 12–14 show the results.

Both MERIT and META show linear scalability with the size of databases from 10 K to 100 K. However, from Figs. 12–14, we find that the linear slop of MERIT seems very close to zero. That is, MERIT is much more scalable than META. As the size of databases grows up, the difference between MERIT and META becomes larger and larger. In a word, MERIT shows much better scalability against the number of items than META.

5.3.2. Against the number of items

Similar to the part of varying database size for mining erasable itemsets, Figs. 15–17 shows the result for running MERIT and META on databases with different number of items. In these experiments, the size of each database is 100 K and the probability distribution of product profit, PD, is N(50, 25). The average product length, APL, of databases used in Figs. 15–17 are 15, 20, and 30 respectively. The thresholds for Figs. 15–17 are 0.025, 0.02 and 0.03, respectively. The number of items in databases for Figs. 15–17 ranges from 100 to 300.

From Figs. 15–17, we find that when the average size of transactions products is low, the running time becomes prone to sudden change for both algorithms. For example, in Fig. 15 where APL is equal to 15, the time-consuming of META algorithm is too large to be measure in a reasonable time. The reason can be explained as follows. With the same item number, database with shorter average size of product may have more erasable itemsets. This has been confirmed in subsection B. Due to the good compactness of NC_sets, when the average size of product changes between 20 and 30, the running time of MERIT grows slowly. However, META increases suddenly when item number rises to 300. Overall, MERIT shows much better scalability against the number of items than META.

6. Conclusions

In this paper, we present a new data representation called NC_sets for storing compressed, crucial information about itemsets of a production database. Based on NC_sets, we develop a new algorithm called MERIT for mining erasable itemsets efficiently. MERIT algorithm not only makes it easy to compute the gain of an itemset via combination operations on NC_sets, but also automatically prune irrelevant data. For evaluating MERIT algorithm, we have conducted extensive experiments on a lot of synthetic product databases. Our performance study shows that the MERIT algorithm is efficient and is on average about 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 investigate techniques to deal with the case that NC_sets cannot be contained in main memory when datasets are too huge. Second, we will extend our algorithm to deal with the case of mining erasable itemsets from distributed databases. Finally, there have been some interesting studies at mining Maximal Frequent (Burdick, Calimlim, Flannick, Gehrké, & Yiü, 2005), closed frequent patterns (Pei, Han, & Mao, 2000; Wang, Han, & Pei, 2003) and top-k frequent patterns (Han, Wang, Lu, & Tzvetkov, 2002; Wang, Han, Lu, & Tzvetkov, 2005) in recent years. Similar to frequent patterns, the extension of erasable itemsets to these special forms is an interesting topic for future research.

Acknowledgements

This work is partially supported by Project 61170091 supported by National Natural Science Foundation of China and Project 2009AA012136 supported by the National High Technology Research and Development Program of China (863 Program).

References


Han, J., Pei, J., & Yin, Y. (2000). Mining frequent patterns without candidate generation. In SIGMOD'00 (pp. 1–12).

Han, J., Wang, J., Lu, Y., & Tzvetkov, P. (2002). Mining top-k frequent closed patterns without minimum support. In ICDE'02 (pp. 211–218).


