Overview

A survey of erasable itemset mining algorithms

Tuong Le,[1,2] Bay Vo[1,2]* and Giang Nguyen[3]

Pattern mining, one of the most important problems in data mining, involves finding existing patterns in data. This article provides a survey of the available literature on a variant of pattern mining, namely erasable itemset (EI) mining. EI mining was first presented in 2009 and META is the first algorithm to solve this problem. Since then, a number of algorithms, such as VME, MERIT, and dMERIT+, have been proposed for mining EI. MEI, proposed in 2014, is currently the best algorithm for mining EIs. In this study, the META, VME, MERIT, dMERIT+, and MEI algorithms are described and compared in terms of mining time and memory usage.

How to cite this article:
WIREs Data Mining Knowl Discov 2014, 4:356–379. doi: 10.1002/widm.1137

INTRODUCTION

Problems related to data mining, including association rule mining,[1–6] applications of association rule mining,[7–9] cluster analysis,[10] and classification,[11–13,55] have attracted research attention. In order to solve these problems, the problem of pattern mining[14] must be first addressed. Frequent itemset mining is the most common problem in pattern mining. Many methods for frequent itemset mining have been proposed, such as Apriori algorithm,[1] FP-tree algorithm,[15] methods based on IT-tree,[5,16] hybrid approaches,[17] and methods for mining frequent itemsets and association rules in incremental datasets.[11,18–24] Studies related to pattern mining include those on frequent closed itemset mining,[25,26] high-utility pattern mining,[27–30] the mining of discriminative and essential frequent patterns,[31] approximate frequent pattern mining,[32] concise representation of frequent itemsets,[33] proportional fault-tolerant frequent itemset mining,[34] frequent pattern mining of uncertain data,[35–39] frequent-weighted itemset mining,[40,41] and erasable itemset (EI) mining.[42–48]

In 2009, Deng et al. defined the problem of EI mining, which is a variant of pattern mining. The problem originates from production planning associated with a factory that produces many types of products. Each product is created from a number of components (items) and creates profit. In order to produce all the products, the factory has to purchase and store these items. In a financial crisis, the factory cannot afford to purchase all the necessary items as usual; therefore, the managers should consider their production plans to ensure the stability of the factory. The problem is to find the itemsets that can be eliminated but do not greatly affect the factory's profit, allowing managers to create a new production plan.

Assume that a factory produces n products. The managers plan new products; however, producing these products requires a financial investment, but the factory does not want to expand the current production. In this situation, the managers can use EI mining to find EIs, and then replace them with the new products while keeping control of the factory's profit. With EI mining, the managers can introduce new products without causing financial instability.

In recent years, several algorithms have been proposed for EI mining, such as META (Mining Erasable iTemsets with the Anti-monotone property),[44] VME (Vertical-format-based algorithm for Mining Erasable itemsets),[45] MERIT (fast Mining ERasable ITemsets),[43] dMERIT+ (using difference

*Correspondence to: vodinhbay@tdt.edu.vn

[1]Division of Data Science, Ton Duc Thang University, Ho Chi Minh City, Vietnam

[2]Faculty of Information Technology, Ton Duc Thang University, Ho Chi Minh City, Vietnam

[3]Faculty of Information Technology, Ho Chi Minh City University of Technology, Ho Chi Minh City, Vietnam

Conflict of interest: The authors have declared no conflicts of interest for this article.

of NC_Set to enhance MERIT algorithm),[47] and MEI (Mining Erasable Itemsets).[46] This study outlines existing algorithms for mining EIs. For each algorithm, its approach is described, an illustrative example is given, and its advantages and disadvantages are discussed. In the experiment section, the performance of the algorithms is compared in terms of mining time and memory usage. Based on the experimental results, suggestions for future research are given.

The rest of this study is organized as follows: Section 2 introduces the theoretical basis of EI mining; Section 3 presents META, VME, MERIT, dMERIT+, and MEI algorithms; Section 4 compares and discusses the runtime and memory usage of these algorithms;Section 5 gives the conclusion and suggestions for future work.

RELATED WORK

Frequent Itemset Mining

Frequent itemset mining[49] is an important problem in data mining. Currently, there are a large number of algorithms that effectively mine frequent itemsets. They can be divided into three main groups:

1. Methods that use a candidate generate-and-test strategy: these methods use a level-wise approach for mining frequent itemsets. First, they generate frequent 1-itemsets which are then used to generate candidate 2-itemsets, and so on, until no more candidates can be generated. Apriori[1] and BitTableFI[50] are two such algorithms.
2. Methods that adopt a divide-and-conquer strategy: these methods compress the dataset into a tree structure and mine frequent itemsets from this tree using a divide-and-conquer strategy. FP-Growth[15] and FP-Growth*[51] are two such algorithms.
3. Methods that use a hybrid approach: these methods use vertical data formats to compress the database and mine frequent itemsets using a divide-and-conquer strategy. Eclat,[2] dEclat,[26] Index-BitTableFI,[52] DBV-FI,[4] and Node-list-based methods[17,53] are some examples.

EI Mining

Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of all items, which are the abstract representations of components of products. A product dataset, *DB*, contains a set of products $\{P_1, P_2, \ldots, P_n\}$. Each product P_i is represented in the form

TABLE 1 | An Example Dataset (DB_e)

Product	Items	Val ($)
P_1	*a, b, c*	2100
P_2	*a, b*	1000
P_3	*a, c*	1000
P_4	*b, c, e*	150
P_5	*b, e*	50
P_6	*c, e*	100
P_7	*c, d, e, f, g*	200
P_8	*d, e, f, h*	100
P_9	*d, f*	50
P_{10}	*b, f, h*	150
P_{11}	*c, f*	100

$\langle Items, Val \rangle$, where *Items* are all items that constitute P_i and *Val* is the profit that the factory obtains by selling product P_i. A set $X \subseteq I$ is called an itemset, and an itemset with k items is called a k-itemset.

The example product dataset in Table 1, DB_e, is used throughout this study, in which $\{a, b, c, d, e, f, g, h\}$ is the set of items (components) used to create all products $\{P_1, P_2, \ldots, P_{11}\}$. For example, P_2 is made from two components, $\{a, b\}$, and the factory earns 1000 dollars by selling this product.

Definition 1. *Let* X (⊆I) *be an itemset. The gain of* X *is defined as:*

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

The gain of itemset X is the sum of profits of the products which include at least one item in itemset X. For example, let $X = \{ab\}$ be an itemset. From DB_e, $\{P_1, P_2, P_3, P_4, P_5, P_{10}\}$ are the products which include {a}, {b}, or {ab} as components. Therefore, $g(X) = P_1 \cdot Val + P_2 \cdot Val + P_3 \cdot Val + P_4 \cdot Val + P_5 \cdot Val + P_{10} \cdot Val = 4450$ dollars.

Definition 2. *Given a threshold* ξ *and a product dataset* DB, *let* T *be the total profit of the factory, computed as:*

$$T = \sum_{P_k \in DB} P_k.Val \tag{2}$$

An itemset X is erasable if and only if:

$$g(X) \leq T \times \xi \tag{3}$$

The total profit of the factory is the sum of profits of all products. From DB_e, $T = 5000$ dollars. An itemset X is called an EI if $g(X) \leq T \times \xi$.

For example, let $\xi = 16\%$. The gain of item h, $g(h) = 250$ dollars. Item h is called an EI with $\xi = 16\%$ because $g(h) = 250 \le 5000 \times 16\% = 800$. This means that the factory does not need to buy and store item h. In that case, the factory will not manufacture products P_8 and P_{10}, but it still has profitability (greater than or equal to $5000*16\% = 4000$ dollars).

EXISTING ALGORITHMS FOR EI MINING

This section introduces existing algorithms for EI mining, namely META,[44] VME,[45] MERIT,[43] dMERIT+,[47] and MEI,[46] which are summarized in Table 2.

TABLE 2 | Summary of Existing Algorithms for Mining EIs

Algorithm	Year	Approach
META	2009	Apriori-like
VME	2010	PID_List-structure-based
MERIT	2012	NC_Set-structure-based
dMERIT+	2013	dNC_Set-structure-based
MEI	2014	dPidset-structure-based

META Algorithm

Algorithm

In 2009, Deng et al. defined EIs, the problem of EI mining, and the META algorithm, an iterative approach that uses a level-wise search for EI mining, which is also adopted by the Apriori algorithm in frequent pattern mining. This approach also uses the property: 'if itemset X is inerasable and Y is a superset of X, then Y must also be inerasable' to reduce the search space. The level-wise-based iterative approach finds erasable $(k+1)$-itemsets by making use of erasable k-itemsets. The details of the level-wise-based iterative approach are as follows. First, the set of erasable 1-itemsets, E_1, is found. Then, E_1 is used to find the set of erasable 2-itemsets E_2, which 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 dataset. The details of META are given in Figure 1.

```
Input: product dataset DB and threshold ξ
Output: EIs, all erasable itemsets in DB
scan DB to get T (the total profit)
E1 = {erasable 1-itemsets in DB}
for (k = 2; Ek-1 ≠ ∅; k++)
   GCk = Gen_Candidate(Ek-1)
   for each product P ∈ DB
      for each candidate itemset C ∈ GCk
         if C ∩ P ≠ ∅ then
            C.val = C.val + P.val
   Ek = {C ∈ GCk | C.val ≤ ξ × T}
return EIs = ∪kEk

Function Gen_Candidate(Ek-1)
candidates = ∅
for each EI A1(={x1, x2 , …xk-2, xk-1})∈ Ek-1
   for each EI A2(={y1, y2, …yk-2, yk-1})∈ Ek-1
      if ((x1=y1)∧(x2=y2)∧…∧(xk-2=yk-2)∧(xk-1 < yk-1)) then
         X = {x1, x2 , …xk-2, xk-1, yk-1}
         if No_Inerasable_Subset(X, Ek-1) then
            add X to Candidates
return Candidates

Function No_Inerasable_Subset(X, Ek-1)
for each (k-1)-subset Xs of X
   if Xs ∉ Ek-1 then
      return FALSE
return TRUE
```

FIGURE 1 | META algorithm.

An Illustrative Example

Consider DB_e with $\xi = 16\%$. First, META determines $T = 5000$ dollars and erasable 1-itemsets $E_1 = \{e, f, d, h, g\}$, with their gains shown in Table 3.

Then, META calls the *Gen_Candidate* function with E_1 as a parameter to create E_2, calls the *Gen_Candidate* function with E_2 as a parameter to create E_3, and calls the *Gen_Candidate* function with E_3 as a parameter to create E_4. E_4 cannot create any EIs of E_5; therefore, META stops. E_2, E_3, and E_4 are shown in Tables 4, 5 and 6, respectively.

TABLE 3 | Erasable 1-Itemsets E_1 and their Gains for DB_e

Erasable 1-itemsets	Val ($)
E	600
F	600
D	350
H	250
G	200

DISCUSSION

The results of META are all EIs. However, the mining time of this algorithm is long because:

1. META scans the dataset the first time to determine the total profit of the factory and n times to determine the information associated with each EI, where n is the maximum level of the result of EIs.
2. To generate candidate itemsets, META uses a naïve strategy, in which an erasable k-itemset X is considered with all remaining erasable k-itemsets used to combine and generate erasable $(k+1)$-itemsets. Only a small number of all remaining erasable k-itemsets which have the same prefix as that of X are combined.

TABLE 4 | Erasable 2-Itemsets E_2 and their Gains for DB_e

Erasable 2-itemsets	Val ($)
Ed	650
Eh	750
Eg	600
Fd	600
Fh	600
Fg	600
Dh	500
Dg	350
Hg	450

For example, consider the erasable 3-itemset {*edh*, *edg*, *fhg*, *fdh*, *fdg*, *fhg*, *dhg*}. META combines the first element {*edh*} with all remaining erasable 3-itemsets {*edg*, *fhg*, *fdh*, *fdg*, *fhg*, *dhg*}. Only {*edg*} is used to combine with {*edh*}, and {*fhg*, *fdh*, *fdg*, *fhg*, *dhg*} are redundant.

VME Algorithm

PID_List Structure

Deng and Xu[45] proposed the VME algorithm for EI mining. This algorithm uses a PID_List (a list

TABLE 5 | Erasable 3-Itemsets E_3 and their Gains for DB_e

Erasable 3-itemsets	Val ($)
Edh	800
Edg	650
Fhg	750
Fdh	600
Fdg	600
Fhg	600
Dhg	500

TABLE 6 | Erasable 4-Itemsets E_4 and their Gains for DB_e

Erasable 4-itemsets	Val ($)
edhg	800
fdhg	600

of product identifiers) structure. The basic concepts associated with this structure are as follows.

Definition 3. *The PID_List of 1-itemset* A ∈ I *is:*

$$PIDs(A) = \bigcup_{\{P_k | A \quad P_k.Items \neq \quad\}} P_k.ID,\ P_k.Val \qquad (4)$$

```
Input: a product dataset DB and threshold ξ
Output: EIs, all erasable itemsets in DB
1.scan DB to get T (the total profit of the factory)
2.scan DB again to find the set of all erasable 1-itemsets, E1, and
their PID_lists
3.for (k = 2; Ek-1 ≠ ∅; k++)
4.    GCk = Gen_Candidate(Ek-1)
5.    Ek = ∅
6.    for each k-itemset P ∈ GCk
7.       compute P.val = ∑_{j=1}^{s} P.PIDs_j.Val //Theorem 2
8.       if P.val ≤ ξ × T then
9.          Ek = Ek ∪ {P}
10.return EI = ∪kEk

function Gen_Candidate(Ek-1)
1.let candidates = ∅
2.for each EI A1(={x1, x2 , …xk-2, xk-1})∈ Ek-1
3.    for each EI A2(={y1, y2, …yk-2, yk-1})∈ Ek-1
4.       if ((x1=y1)∧(x2=y2)∧…∧(xk-2=yk-2)∧(xk-1 < yk-1)) then
5.          X = {x1, x2, …xk-2, xk-1, yk-1}
6.          if No_Unerasable_Subset(X, Ek-1) then
7.             X.PID_list = A1.PID_list ∪ A2.PID_list //Theorem 1
8.             add {(X, X.PID_list)} to Candidates
9.return Candidates

function No_Unerasable_Subset(X, Ek-1)
1.for each (k-1)-subset Xs of X
2.    if Xs ∉ Ek-1 then
3.       return False
4.return True
```

FIGURE 2 | VME algorithm.

TABLE 7 | Erasable 1-Itemsets E_1 and their PID_Lists for DB_e

Erasable 1-itemsets	PID_Lists
e	⟨4, 150⟩, ⟨5, 50⟩, ⟨6, 100⟩, ⟨7, 200⟩, ⟨8, 100⟩
f	⟨7, 200⟩, ⟨8, 100⟩, ⟨9, 50⟩, ⟨10, 150⟩, ⟨11, 100⟩
d	⟨7, 200⟩, ⟨8, 100⟩, ⟨9, 50⟩
h	⟨8, 100⟩, ⟨10, 150⟩
g	⟨7, 200⟩

Example 1. *Considering* DB_e, PIDs(d) = {⟨*7, 200*⟩, ⟨*8, 100*⟩, ⟨*9, 50*⟩} *and* PIDs(h) = {⟨*8, 100*⟩, ⟨*10, 150*⟩}.

Theorem 1. *Let* XA *and* XB *be two erasable* k*-itemsets. Assume that* PIDs(XA) *and* PIDs(XB) *are PID_Lists associated with* XA *and* XB*, respectively. The PID_List of* XAB *is determined as follows:*

$$PIDs\,(XAB) = PIDs\,(XA)\quad PIDs\,(XB) \qquad (5)$$

Example 2. *According to Example 1 and Theorem 1,* PIDs(dh) = PIDs(d) ∪ PIDs(h) = {⟨*7, 200*⟩, ⟨*8, 100*⟩, ⟨*9, 50*⟩} ∪ {⟨*8, 100*⟩, ⟨*10, 150*⟩} = {⟨*7, 200*⟩, ⟨*8, 100*⟩, ⟨*9, 50*⟩, ⟨*10, 150*⟩}.

Theorem 2. *The gain of an itemset,* X, *can be computed as follows:*

$$g\,(X) = \sum_{j=1}^{n} PIDs_j.Val \qquad (6)$$

Example 3. *According to Example 2 and Theorem 2,* PIDs(dh) = {⟨*7, 200*⟩, ⟨*8, 100*⟩, ⟨*9, 50*⟩, ⟨*10, 150*⟩}; *therefore,* g(dh) = *200 + 100 + 50 + 150 = 500 dollars.*

Mining EIs Using PID_List Structure

Based on Definition 3, Theorem 1, and Theorem 2, Deng and Xu[45] proposed the VME algorithm for EI mining, shown in Figure 2.

An Illustrative Example

Consider DB_e with $\xi = 16\%$. First, VME determines $T = 5000$ dollars and erasable 1-itemsets $E_1 = \{e, f, d, h, g\}$, with their PID_Lists shown in Table 7.

Second, VME uses E_1 to create E_2, E_2 to create E_3, and E_3 to create E_4. E_4 does not create any EIs; therefore, VME stops. E_2, E_3, and E_4 are shown in Tables 8, 9 and 10, respectively.

DISCUSSION

VME is faster than META. However, some weaknesses associated with VME are:

TABLE 8 | Erasable 2-Itemsets E_2 and their PID_Lists for DB_e

Erasable 2-itemsets	PID_Lists
ed	⟨4, 150⟩, ⟨5, 50⟩, ⟨6, 100⟩, ⟨7, 200⟩, ⟨8, 100⟩, ⟨9, 50⟩
eh	⟨4, 150⟩, ⟨5, 50⟩, ⟨6, 100⟩, ⟨7, 200⟩, ⟨8, 100⟩, ⟨10, 150⟩
eg	⟨4, 150⟩, ⟨5, 50⟩, ⟨6, 100⟩, ⟨7, 200⟩, ⟨8, 100⟩,
fd	⟨7, 200⟩, ⟨8, 100⟩, ⟨9, 50⟩, ⟨10, 150⟩, ⟨11, 100⟩
fh	⟨7, 200⟩, ⟨8, 100⟩, ⟨9, 50⟩, ⟨10, 150⟩, ⟨11, 100⟩
fg	⟨7, 200⟩, ⟨8, 100⟩, ⟨9, 50⟩, ⟨10, 150⟩, ⟨11, 100⟩
dh	⟨7, 200⟩, ⟨8, 100⟩, ⟨9, 50⟩, ⟨10, 150⟩
dg	⟨7, 200⟩, ⟨8, 100⟩, ⟨9, 50⟩
hg	⟨7, 200⟩, ⟨8, 100⟩, ⟨10, 150⟩

TABLE 9 | Erasable 3-Itemsets E_3 and their PID_Lists DB_e

Erasable 3-itemset	PID_Lists
edh	⟨4, 150⟩, ⟨5, 50⟩, ⟨6, 100⟩, ⟨7, 200⟩, ⟨8, 100⟩, ⟨9, 50⟩, ⟨10, 150⟩
edg	⟨4, 150⟩, ⟨5, 50⟩, ⟨6, 100⟩, ⟨7, 200⟩, ⟨8, 100⟩, ⟨9, 50⟩
fhg	⟨7, 200⟩, ⟨8, 100⟩, ⟨9, 50⟩, ⟨10, 150⟩, ⟨11, 100⟩
fdh	⟨7, 200⟩, ⟨8, 100⟩, ⟨9, 50⟩, ⟨10, 150⟩, ⟨11, 100⟩
fdg	⟨7, 200⟩, ⟨8, 100⟩, ⟨9, 50⟩, ⟨10, 150⟩, ⟨11, 100⟩
fhg	⟨7, 200⟩, ⟨8, 100⟩, ⟨9, 50⟩, ⟨10, 150⟩, ⟨11, 100⟩
dhg	⟨7, 200⟩, ⟨8, 100⟩, ⟨9, 50⟩, ⟨10, 150⟩

TABLE 10 | Erasable 4-Itemsets E_4 and their PID_Lists DB_e

Erasable 4-itemset	PID_Lists
edhg	⟨4, 150⟩, ⟨5, 50⟩, ⟨6, 100⟩, ⟨7, 200⟩, ⟨8, 100⟩, ⟨9, 50⟩, ⟨10, 150⟩
fdhg	⟨7, 200⟩, ⟨8, 100⟩, ⟨9, 50⟩, ⟨10, 150⟩, ⟨11, 100⟩

1. VME scans the dataset to determine the total profit of the factory and then scans the dataset again to find all erasable 1-itemsets and their PID_Lists. Scanning the dataset takes a lot of time and memory. The dataset can be scanned once only if carefully considered.
2. VME uses the breadth-first-search strategy, in which all erasable $(k-1)$-itemsets are used to create erasable k-itemsets. Nevertheless, classifying erasable $(k-1)$-itemsets with the same prefix as that of erasable $(k-2)$-itemsets

```
procedure Construct_WPPC_tree(DB, ξ)
1. scan DB once to find E1, their gains, their frequency, and the
total gain of the factory (T)
2. sort E1 in frequency descending order
3. create H1, the hash table of E1
4. create the root of a WPPC-tree, ℛ, and label it as 'null'
5. for each P ∈ DB do
6.   remove inerasable 1-itemsets
7.   sort its erasable 1-itemsets in frequency descending order
8.   Insert_tree(P, ℛ)
9. scan WPPC-tree to generate the pre-order and post-order number
for each node and H1
10. return ℛ, E1, H1, and T

procedure Insert_tree(P, ℛ)
1. while (P is not null) do
2.   P1 ← the first items of P
3.   P ← P \ P1
4.   if ℛ has a child N such that N.item-name = P1 then
5.     N.Weight = N.Weight + P.Val
6.   else
7.     create a new node N
8.     N.Weight ← P.Val
9.     ℛ.Childnodes = N
10.  Insert_tree(P, N)
11. end while
```

FIGURE 3 | WPPC-tree construction algorithm.

takes a lot of time and operations. For example, the erasable 2-itemsets are {*ed*, *eh*, *eg*, *fd*, *fh*, *fg*, *dh*, *dg*, *hg*}, which have four 1-itemset prefixes, namely {*e*}, {*f*}, {*d*}, and {*h*}. The algorithm divides the elements into groups of erasable 2-itemsets, which have the same prefix as that of erasable 1-itemsets. In particular, the erasable 2-itemsets are classified into four groups: {*ed*, *eh*, *eg*}, {*fd*, *fh*, *fg*}, {*dh*, *dg*}, and {*hg*}. Then, the algorithm combines the elements of each group to create the candidates of erasable 3-itemsets, which are {*edh*, *edg*, *fhg*, *fdh*, *fdg*, *fhg*, *dhg*}.

3. VME uses the union strategy, in which X's PID_List is a subset of Y's PID_List if $X \subset Y$. This strategy requires a lot of memory and operations for a large number of EIs.
4. VME stores each product's profit (*Val*) in a pair ⟨*PID*, *Val*⟩ in PID_List. This leads to data duplication because a pair ⟨*PID*, *Val*⟩ can appear in many PID_Lists. Therefore, this algorithm requires a lot of memory. Memory usage can be reduced by using an index of gain.

MERIT Algorithm

Deng and Wang,[54] and Deng et al.[53] presented the WPPC-tree, an FP-tree-like structure. Then, the authors created the N-list structure based on WPPC-tree. Based on this idea, Deng et al.[43] proposed the NC_Set structure for fast mining EIs.

TABLE 11 | DB_e after Removal of 1-Itemsets ($\xi = 16\%$) Which are Not Erasable and Sorting of Remaining Erasable 1-Itemsets in Ascending Order of Frequency

Product	Items	Val ($)
P_4	*e*	150
P_5	*e*	50
P_6	*e*	100
P_7	*e*, *f*, *d*, *g*	200
P_8	*e*, *f*, *d*, *h*	100
P_9	*f*, *d*	50
P_{10}	*f*, *h*	150
P_{11}	*f*	100

WPPC-tree

Definition 4. *(WPPC-tree) A WPPC-tree, ℛ, is a tree where the information stored at each node comprises tuples of the form:*

$$\left\langle N_i.\textit{item-name}, N_i.\textit{weight}, N_i.\textit{childnodes}, \times N_i.\textit{pre-order}, N_i.\textit{post-order}\right\rangle \quad (7)$$

where N_i·item-name *is the item identifier,* N_i·weight *and* N_i·childnodes *are the gain value and set of child nodes associated with the item, respectively,* N_i·pre-order *is the order number of the node when the tree is traversed top-down from left to right, and* N_i·post-order *is the order number of the node*

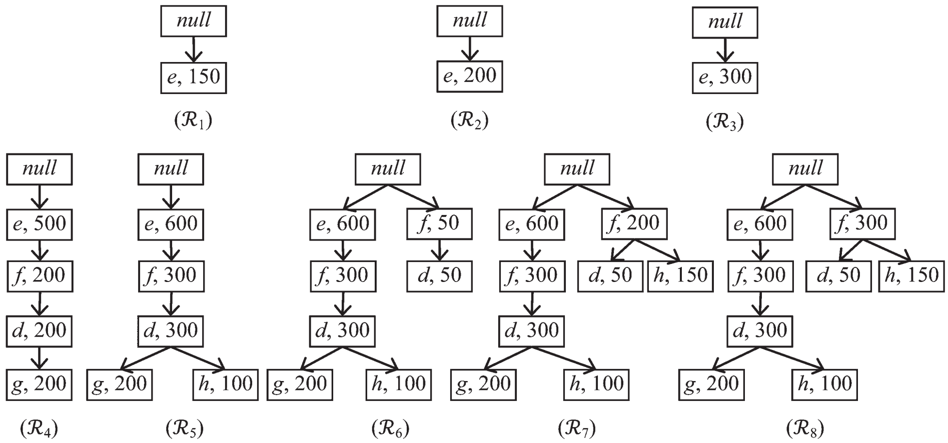

FIGURE 4 | Illustration of WPPC-tree construction process for *DBe*.

when the tree is traversed bottom-up from left to right.

Deng and Xu[43] proposed the WPPC-tree construction algorithm to create a WPPC-tree shown in Figure 3.

Consider DB_e with $\xi = 16\%$. First, the algorithm scans the dataset to find the erasable 1-itemsets (E_1). The algorithm then scans the dataset again and, for each product, removes the inerasable 1-itemsets. The remaining 1-itemsets are sorted in ascending order of frequency, as shown in Table 11 (where P_1 is removed because it has no erasable 1-itemsets).

These itemsets are then used to construct a WPPC-tree by inserting each item associated with each product into the tree. Given the nine remaining products, P_4–P_{11}, the tree is constructed in eight steps, as shown in Figure 4. Note that in Figure 4 (apart from the root node), each node N_i represents an item in I and each is labeled with the item identifier (N_i·*item-name*) and the item's gain value (N_i·*weight*).

Finally, the algorithm traverses the WPPC-tree to generate the *pre-order* and *post-order* numbers to give a WPPC-tree of the form shown in Figure 5, where each node N_i has been annotated with its *pre-order* and *post-order* numbers (N_i·*pre-order* and N_i·*post-order*, respectively).

NC_Set Structure

Definition 5. *(node code) The node code of a node* N_i *in the WPPC-tree, denoted by* C_i, *is a tuple of the form:*

$$C_i = \langle N_i.pre\text{-}order, N_i.post\text{-}order : N_i.weight \rangle \quad (8)$$

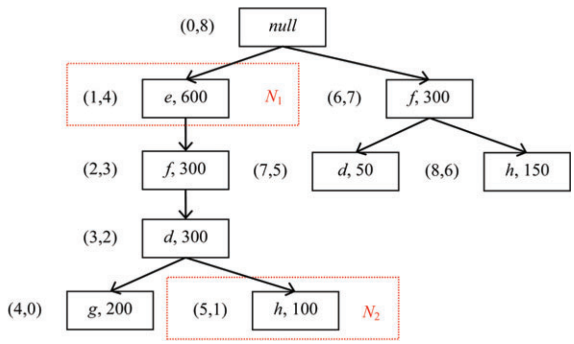

FIGURE 5 | WPPC-tree for *DBe* with $\xi = 16\%$.

Theorem 3. *A node code* C_i *is an ancestor of another node code* C_j *if and only if* C_i·pre-order $\leq C_j$·pre-order *and* C_i·post-order $\geq C_j$·post-order.

Example 4. *In Figure 5, the node code of the highlighted node* N_1 *is* $\langle 1,4{:}600 \rangle$, *in which* N_1·pre-order = 1, N_1·post-order = 4, *and* N_1·weight = 600; *and the node code of* N_2 *is* $\langle 5,1{:}100 \rangle$. N_1 *is an ancestor of* N_2 *because* N_1·pre-order = 1 < N_2·pre-order = 5 *and* N_1·post-order = 4 > N_2·post-order = 1.

Definition 6. *(NC_Set of an erasable 1-itemset) Given a WPPC-tree* $\mathcal{R}$ *and a 1-itemset* A, *the NC_Set of* A, *denoted by* NCs(A), *is the set of node codes in* $\mathcal{R}$ *associated with* A *sorted in descending order of* C_i·pre-order.

$$NCs\,(A) = \cup_{\{\forall N_i \in \mathcal{R},\ N_i.item-name=A\}} C_i \quad (9)$$

where C_i *is the node code of* N_i.

Example 5. *According to* $\mathcal{R}$ *in Figure 5,* NCs(e) = {⟨1,4:600⟩}, NCs(h) = {⟨5,1:100⟩, ⟨8,6:150⟩} *and* NCs(d) = {⟨3,2:300⟩, ⟨7,5:50⟩}.

Definition 7. *(complement of a node code set) Let* XA *and* XB *be two EIs with the same prefix* X *(*X *can* *be an empty set). Assume that* A *is before* B *with respect to* E_1 *(the list of identified erasable 1-itemsets ordered according to ascending order of frequency).* NCs(XA*) and* NCs(XB*) are the NC_Sets of* XA *and* XB*, respectively. The complement of one node code set with respect to another is defined as follows:*

$$NCs(XB) \setminus NCs(XA) = NCs(XB) \setminus \{C_j \in NCs(XB) \mid \exists C_i \in NCs(XA), \times C_i \text{ is an ancestor } C_j\} \quad (10)$$

Example 6. NCs(h*)*\NCs*(e) = {⟨5,1:100⟩, ⟨8,6:150⟩} \{⟨1,4:600⟩} = {⟨5,1:100⟩, ⟨8,6:150⟩}\{⟨5,1:100⟩} = {⟨8, 6:150⟩}. Similarly,* NC(d)\NC*(e) = {⟨3,2:300⟩, ⟨7,5: 50⟩}\{⟨1,4:600⟩} = {⟨7,5:50⟩}.*

Definition 8. *(NC_Set of erasable k-itemset) Let* XA *and* XB *be two EIs with the same prefix* X. NCs(XA*) and* NCs(XB*) are the NC_Sets of* XA *and* XB*, respectively. The NC_Set of* XAB *is determined as:*

$$NCs(XAB) = NCs(XA) \cup [NCs(XB) \setminus NCs(XA)] \quad (11)$$

Example 7. *According to* Example 6 *and Definition 8, the NC_Set of* eh *is* NCs(eh*)* = NCs(*e)* ∪ *[*NCs(h*)*\NCs*(e)] = {⟨1,4:600⟩} ∪ {⟨8,6:150⟩} = {⟨1,4: 600⟩, ⟨8,6:150⟩} and the NC_Set of* ed *is* NCs(ed*) = {⟨1,4:600⟩, ⟨7,5:50⟩}. Similarly, the NC_Set of* edh *is* NCs(edh*)* = NCs(ed*)* ∪ *[*NCs(eh*)*\NCs(ed*)] = {⟨1,4:600⟩, ⟨7,5:50⟩}* ∪ *[{⟨1,4:600⟩, ⟨8,6:150⟩}\{⟨1,4: 600⟩, ⟨7,5:50⟩}] = {⟨1,4:600⟩, ⟨7,5:50⟩, ⟨8,6:150⟩}.*

Theorem 4. *Let* X *be an itemset and* NCs(X*) be the NC_Set of* X*. The gain of* X *is computed as follows:*

$$g(X) = \sum_{C_i \in NCs(X)} C_i.weight \quad (12)$$

Example 8. *Based on* Example 7*, the NC_Set* edh *is* NCs(edh*) = {⟨1,4:600⟩, ⟨7,5:50⟩, ⟨8,6:150⟩}. Therefore, the gain of* edh *is* g(edh*) = 600 + 50 + 150 = 800 dollars.*

Efficient Method for Combining Two NC_Sets

To speed up the runtime of EI mining, Deng and Xu[43] proposed an efficient method for combining two NC_Sets, shown in Figure 6.

Mining EIs Using NC_Set Structure

Based on the above theoretical background, Deng and Xu[43] proposed an efficient algorithm for mining EIs, called MERIT, shown in Figure 7.

MERIT+ Algorithm

Algorithm

MERIT has some problems which cause the loss of a large number of EIs:

1. MERIT uses an 'if' statement to check all subsets of $(k-1)$-itemsets of a k-itemset X to determine whether they are erasable to avoid executing the procedure *NC_Combination*. However, MERIT uses the deep-first-search strategy so there are not enough $(k-1)$-itemsets in the results for this check. The 'if' statement is always false, so all erasable k-itemsets $(k > 2)$ are always inerasable. The results of MERIT are thus erasable 1-itemsets and erasable 2-itemsets. Once X's NC_Set is determined, the algorithm can immediately decide whether X is erasable. Hence, the if statement in this algorithm is unnecessary.
2. MERIT enlarges the equivalence classes of $EC_v[k]$; therefore, the results of the algorithm are not all EIs. This improves the mining time, but not all EIs are mined.

Le et al.[46,47] thus introduced a revised algorithm called MERIT+, derived from MERIT, that is capable of mining all EIs but does not: (1) check all subsets of $(k-1)$-itemsets of a k-itemset X to determine whether they are erasable and (2) enlarge the equivalence classes.

An Illustrative Example

To explain MERIT+, the process of the MERIT+ algorithm for DB_e with $\xi = 16\%$ is described below. First, MERIT+ uses the WPPC-tree construction algorithm shown in Figure 3 to create the WPPC-tree (Figure 5). Next, MERIT+ scans this tree to generate the NC_Sets associated with erasable 1-itemsets. Figure 8 shows E_1 and its NC_Set.

Then, MERIT+ uses the divide-and-conquer strategy for mining EIs. The result of this algorithm is shown in Figure 9.

DISCUSSION

MERIT+ and MERIT still have three weaknesses:

1. They use the union strategy in which $NCs(X) \subset NCs(Y)$ if $X \subset Y$. As a result, their memory usage is large for a large number of EIs.
2. They scan the dataset three times to build the WPPC-tree. Then, they scan the WPPC-tree twice to create the NC_Set of erasable 1-itemsets. The previous steps take a lot of time and operations.
3. They store the value of a product's profit in each NC of NC_Set, which leads to data duplication.


```
function NC_Combination(NL1, NL2)
1. let NL ← ∅ and c ← 0
2. for j = 1 to |NL1| do
3.   while c ≤ |NL2| && NL1[j].pre-order > NL2[c].pre-order do
4.     insert NL2[c] into NL
5.     c++
6.   insert NL1[j] into NL
7.   while c ≤ |NL2| && NL1[j].pre-order ≥ NL2[c].pre-order do
8.     c++
9. while c < |NL2| do
10.  insert NL2[c] into NL
11.return NL
```

FIGURE 6 | Efficient method for combining two NC_Sets.

```
Input: the product dataset DB and threshold ξ.
Output: EIs, the set of all erasable itemsets.
1. call Construct_WPPC_tree(DB, ξ) to generate the tree, WPPC, and
the set of erasable 1-itemsets, E1
2. EIs ← E1
3. scan WPPC to generate the NC_Sets associated with E1
4. if |E1| > 1 then
5.    mining_E(E1)
6. reture E

procedure mining_E(ECv)
1. for k ← 0 to |ECv| do
2.    ECnext ← ∅
3.    for j ← k+1 to |ECv| do
4.      Cd ← ECv[k] ∪ ECv[j]
5.      if any subset of Cd with length |Cd|-1 is erasable then
6.        Cd.NC_Set ← NC_Combination(ECv[k].NC_Set, ECv[j].NC_Set)
7.        if Cd.Gain ≤ T × ξ then
8.          add Cd to ECnext
9.    scan NCnext to find all itemsets in ECnext whose NC_Set is equal
to the NC_Set of ECv[k], and use these itemsets to enlarge the
equivalence class ECv[k], and also remove them from ECnext
10.   update equivalence classes of ECnext due to the change of ECv[k]
11.   if |ECnext| > 1 then
12.     mining_E(ECnext)
13.EIs = EIs ∪ ECv
```

FIGURE 7 | MERIT algorithm.

dMERIT+ Algorithm

Index of Weight

Definition 9. *(index of weight) Let $\mathcal{R}$ be a WPPC-tree. The index of weight is defined as:*

$$W\left[N_i pre\right] = N_i weight \tag{13}$$

where $N_i \in \mathcal{R}$ *is a node in* $\mathcal{R}$.

The index of weight for $\mathcal{R}$ shown in Figure 5 is presented in Table 12. Note that the index for node N_i is equivalent to its pre-order number (N_i·*pre-order*). Using the index of weight, a new node code structure ⟨N_i·*pre-order*, N_i·*post-order*⟩, called NC′, and a new NC_Set format (NC′_Set) are proposed (Le et al., 2013). NC′ and NC′_Set make the dMERIT+ algorithm efficient by reducing the memory requirements and speeding up the weight acquisition process for individual nodes.

Example 9. *Consider the following:*

1. *In* Example 8, NCs(edh) = {⟨1,4:600⟩, ⟨7,5:50⟩, ⟨8,6:150⟩}. *Therefore,* g(edh) = 600 + 50 + 150 = 800 *dollars.*
2. *The NC′_Set of* edh *is* NC′s(edh) = {⟨1,4⟩, ⟨7,5⟩, ⟨8,6⟩}. *From this NC′_Set, the dMERIT+ algorithm can easily determine the gain of* edh *by using the index of weight as follows:* g(edh) = W[1] + W[7] + W[8] = 600 + 50 + 150 = 800 *dollars.*

Example 9 *shows that using NC′_Sets lowers the memory requirement for NC′ compared to that for NC_Sets.*

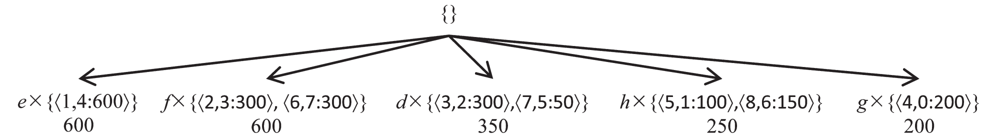

FIGURE 8 | Erasable 1-itemsets, E1, and its NC_Sets for *DBe* with $\xi = 16\%$.

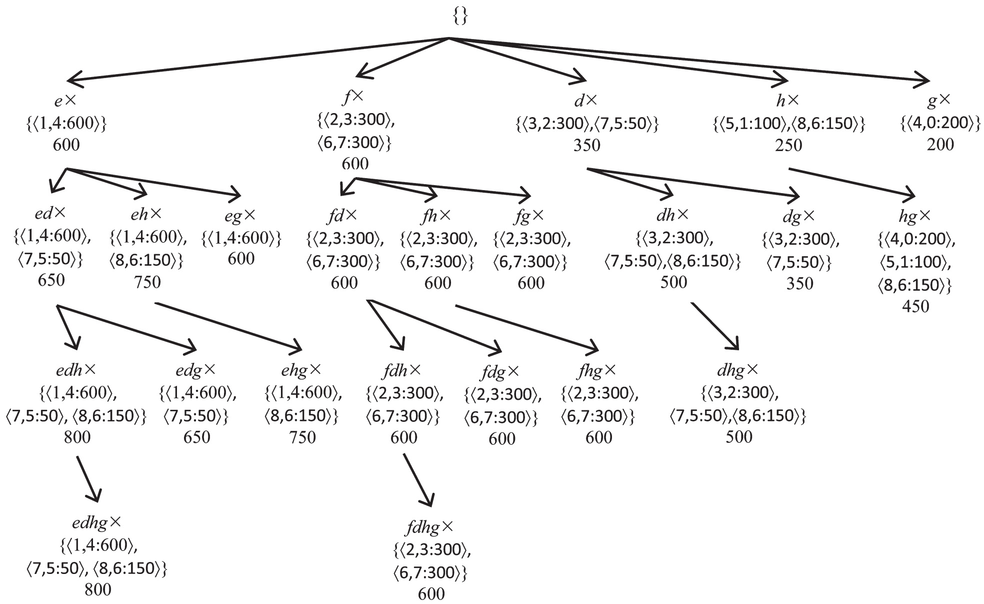

FIGURE 9 | Result of MERIT+ for *DBe* with $\xi = 16\%$.

dNC′_Set Structure

Definition 10. *(dNC′_Set) Let* XA *with its NC′_Set,* NC′s*(*XA*), and* XB *with its NC′_Set,* NC′s*(*XB*), be two itemsets with the same prefix* X *(*X *can be an empty set). The difference NC′_Set of* NC′s*(*XA*) and* NC′s*(*XB*), denoted by* dNC′s*(*XAB*), is defined as follows:*

$$dNC's(XAB) = NC's(XB) \setminus NC's(XA) \qquad (14)$$

Example 10. dNC′s*(*eh*) =* NC′s*(*h*)*NC′s*(*e*) = {⟨5,1⟩, ⟨8,6⟩}\{⟨1,4⟩} = {⟨8,6⟩} and* dNC′s*(*ed*) =* NC′s*(*d*)*NC′s*(*e*) = {⟨7,5⟩}.*

Theorem 5. *Let* XA *with its dNC′_Set,* dNC's*(*XA*), and* XB *with its dNC′_Set,* dNC's*(*XB*), be two itemsets with the same prefix* X *(*X *can be an empty set). The dNC′_Set of* XAB *can be computed as:*

$$dNC's(XAB) = dNC's(XB) \setminus dNC's(XA) \qquad (15)$$

Example 11. *According to* Example 7, NC′s*(*eh*) = {⟨1,4⟩, ⟨8,6⟩} and* NC′s*(*ed*) = {⟨1,4⟩, ⟨7,5⟩}. Therefore,* dNC′s*(*edh*) =* NC′s*(*eh*)*NC′s*(*ed*) = {⟨1,4⟩, ⟨8,6⟩}\{⟨1,4⟩, ⟨7,5⟩} = {⟨8,6⟩}.*

According to Example 10, dNC′s*(*eh*) =* NC′s*(*h*)* NC′s*(*e*) = {⟨8,6⟩} and* dNC′s*(*ed*) =* NC′s*(*d*)* NC′s *(*e*) = {⟨7,5⟩}. Therefore,* dNC′s*(*edh*) =* dNC′s*(*eh*)* dNC′s*(*ed*) = {⟨8,6⟩} {⟨7,5⟩} = {⟨8,6⟩}.*

From (1) and (2), dNC′s*(*edh*) = {⟨8,6⟩}. Therefore, Theorem 5 is verified through this example.*

Theorem 6. *Let the gain (weight) of* XA *be* g*(*XA*). Then, the gain of* XAB, g*(*XAB*), is computed as follows:*

$$g(XAB) = g(XA) + \sum_{C_i \in \, dNC'(XAB)} W\left[C_i.pre\right] \qquad (16)$$

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

Example 12. *Consider the following:*

1. *According to* Example 8, g*(*edh*) = 800 dollars.*
2. NC′s*(*e*) = {⟨1,4⟩},* NC′s*(*d*) = {⟨3,2⟩, ⟨7,5⟩} and* NC′s*(*h*) = {⟨5,1⟩, ⟨8,6⟩}. Therefore,* g*(*e*) = 600,* g*(*d*) = 350 and* g*(*h*) = 250.*

– According to Example 10, dNC′s*(*ed*) = {⟨7,5⟩} and* dNC′s*(*eh*) = {⟨8,6⟩}. Therefore,* g*(*ed*) =* g*(*e*) +*

TABLE 12 | Index of Weight for DB_e with $\xi = 16\%$

Pre-order	1	2	3	4	5	6	7	8
Weight	600	300	300	200	100	300	50	150

W*[7] = 600 + 50 = 650 dollars and* g(eh) *=*g(e) *+* W*[8] = 600 + 150 = 750 dollars.*

– According to Example 11, dNC′s*(*edh*) = {⟨8,6⟩}. Therefore,* g(edh) = g(ed) + W*[8] = 650 + 150 = 800 dollars.*

From (1) and (2), g(edh) *= 800 dollars. Therefore, Theorem 6 is verified through this example.*

Theorem 7. *Let* XA *with its NC′_Set,* NC's*(*XA*), and* XB *with its NC′_Set,* NC's*(*XB*), be two itemsets with the same prefix* X*. Then:*

$$dNC's\,(XAB) \subset NC's\,(XAB) \qquad (17)$$

Example 13. *Consider the following:*

1. *Based on* Example 7, *the NC′_Set of* edh *is* NC′s*(*edh*) = {⟨1,4⟩, ⟨7,5⟩, ⟨8,6⟩}.*
2. *Based on* Example 11, *the dNC′_Set of* edh *is* dNC′s*(*edh*) = {⟨8,6⟩}.*

Obviously, dNC′s*(*edh*) = {⟨8,6⟩} ⊂* NC′s*(*edh*) = {⟨1,4⟩, ⟨7,5⟩, ⟨8,6⟩}. Therefore, Theorem 7 is verified through this example.*

With an itemset XAB*, Theorem 7 shows that using a dNC′_Set is always better than using an NC′_Set. The dMERIT+ algorithm requires less memory and has a faster runtime than those of MERIT+ because there are fewer elements in a dNC′_Set than in an NC'_Set.*

Efficient Method for Subtracting Two NC'_Sets

To speed up the runtime of EI mining, Le et al.[47] proposed an efficient method for determining the difference NC'_Set of two dNC'_Sets, shown in Figure 10.

Mining EIs Using dNC'_Set Structure

Based on the above theoretical background, Le et al.[47] proposed the dMERIT+ algorithm, shown in Figure 11.

An Illustrative Example

Consider DB_e with ξ = 16%. First, dMERIT+ calls the WPPC-tree construction algorithm presented in Figure 3 to create the WPPC_tree, $\mathscr{R}$ (see Figure 5), and then identifies the erasable 1-itemsets E_1 and the total gain for the factory T. The *Generate_NC′_Sets*

```
function dNC'_Set(NC1, NC2)
NC3 ← ∅ and Gain ← 0
let i = 0 and j = 0
while i < NC1.size and j < NC2.size do
  if NC1[i].pre-order ≤ NC2[j].pre-order then
    if NC1[i].post-order ≤ NC2[j].post-order then
      j++
    else
      i++
  else
    add NC2[j] to NC3
    Gain = Gain + W[NC2[j].pre-order]
    j++
while j < NC2.size do
    add NC2[j] to NC3
    Gain = Gain + W[NC2[j].pre-order]
    j++
return NC3 and Gain
```

FIGURE 10 | Efficient method for subtracting two dNC′_Sets.

procedure is then used to create NC′_Sets associated with E_1 (see Figure 12).

The *Mining_E* procedure is then called with E_1 as a parameter. The first erasable 1-itemset {*e*} is combined in turn with the remaining erasable 1-itemsets {*f*, *d*, *h*, *g*} to create the 2-itemset child nodes: {*ef*, *ed*, *eh*, *eg*}. However, {*ed*} is excluded because $g(\{ef\}) = 900 > T \times \xi = 800$ dollars. Therefore, the erasable 2-itemsets of node {*e*} are {*ed*, *eh*, *eg*} (Figure 13).

The algorithm adds {*ed*, *eh*, *eg*} to the results and uses them to call the *Mining_E* procedure to create the erasable 3-itemset descendants of node {*e*}. The first of these, {*ed*}, is combined in turn with the remaining elements {*eh*, *eg*} to produce the erasable 3-itemsets {*edh*, *edg*}. Next, the erasable 3-itemsets of node {*ed*} are used to create erasable 4-itemset {*edhg*}. Similarity, the node {*eh*}, the second element of the set of erasable 2-itemset child nodes of {*e*}, is combined in turn with the remaining elements to give {*ehg*}. The erasable 3-itemset descendants of node {*e*} are shown in Figure 14.

The algorithm continues in this manner until all potential descendants of the set of erasable 1-itemsets have been considered. The result is shown in Figure 15.

When considering the memory usage associated with the MERIT+ and dMERIT+ algorithms, the following can be observed:

1. The memory usage can be determined by summing either: (a) the memory required to

```
Input: A product dataset DB and a threshold ξ
Output: EIs, the set of all erasable itemsets
1. Construct_WPPC_tree(DB, ξ) to generate R, E1, H1, and T
2. Generate_NC'_Sets(R, E1)
3. EIs ← E1
4. if E1.size > 1 then
5.   Mining_ E(E1)
6. return EIs

procedure Generate_NC'_Sets(R, E1)
1. NC1 ← R.post-order and R.pre-order2.pos = H1[R.item-name]
2. add NC1 to E1[pos].NC'_Set
3. for each Child in R.ChildNodes do
4.   Generate_NC'_Sets(Child)

procedure Mining_E(EC)
1. for k ← 1 to EC.size do
2.   ECnext ← ∅
3.   for j ← (k+1) to EC.size do
4.     let e1 and e2 be the last item of EC[k].Items and EC[j].Items
respectively
5.     if H1[e1] < H1[e2] then
6.       EI.Items ← EC[k].Items + {e2}
7.       (EI.NC'_Set and Gain) ← dNC'_Set(EC[k].NC'_Set,
EC[j].NC'_Set)
8.       EI.Gain = EC[k].Gain + Gain
9.     else
10.      EI.Items ← EC[j].Items + {e1}
11.      (EI.NC'_Set and Gain) ← dNC'_Set(EC[j].NC'_Set,
EC[k].NC'_Set)
12.      EI.Gain = EC[j].Gain + Gain
13.    if EI.Gain ≤ ξ × T then
14.      add EI to ECnext
15.      add EI to EIs
16.    if ECnext.size > 1 then
17.      Mining_E(ECnext)
```

FIGURE 11 | dMERIT+ algorithm.

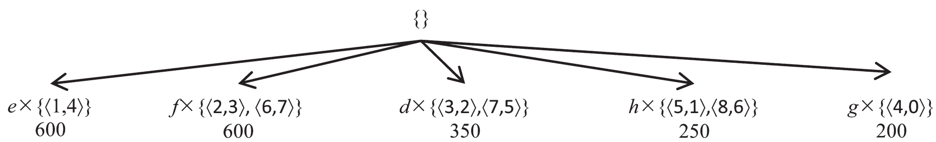

FIGURE 12 | Erasable 1-itemsets and their NC′_Set for *DBe* with $\xi = 16\%$.

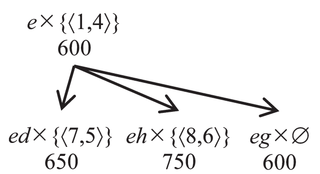

FIGURE 13 | Erasable 2-itemsets of node {e} for *DBe* with $\xi = 16\%$.

store EIs, their dNC′_Sets, and the index of weight (dMERIT+ algorithm) or (b) the memory required to store EIs and their NC_Sets (MERIT+ algorithm).

2. N_i·*pre-order*, N_i·*post-order*, N_i·*weight*, the item identifier, and the gain of an EI are represented in an integer format, which requires 4 bytes in memory.

The number of items included in dMERIT+'s output (see Figure 15) is 101. In addition, dMERIT+ also requires an array with eight elements as the index of weight. Therefore, the memory usage required by dMERIT+ is $(101+8) \times 4 = 436$ bytes. For the MERIT+ algorithm, the number of EIs and the number of associated NC_Sets (see Figure 9) is 219. Hence, the memory usage required by MERIT is $219 \times 4 = 876$ bytes. Thus, this example shows that the memory usage for dMERIT+ is less than that for MERIT+.

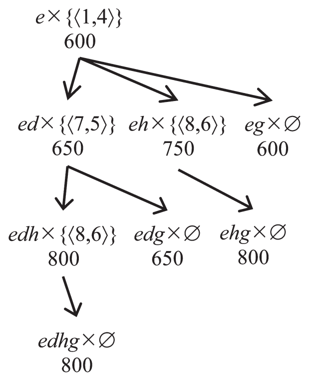

FIGURE 14 | EIs of node {*e*} for *DBe* with $\xi = 16\%$.

MEI Algorithm

Index of Gain

Definition 11. *(index of gain) Let* DB *be the product dataset. An array is defined as the index of gain as:*

$$G[i] = P_i.Val \tag{18}$$

where $P_i \in$ DB *for* $1 \leq i \leq n$.

According to Definition 11, the gain of a product P_i is the value of the element at position i in the index of gain.

For DB_e, the index of gain is shown in Table 13. For example, the gain of product P_4 is the value of the element at position 4 in G denoted by G[4] = 150 dollars.

Pidset—The Set of Product Identifiers

Definition 12. *(pidset) For an itemset* X*, the set of product identifiers* p(X) *is denoted as follows:*

$$p(X) = \bigcup_{A \in X} p(A) \tag{19}$$

where A *is an item in* X *and* p(A) *is the pidset of item* A*, i.e., the set of product identifiers which includes* A.

Definition 13. *(gain of an itemset based on pidset) Let* X *be an itemset. The gain of* X *denoted by* g(X) *is computed as follows:*

$$g(X) = \sum_{P_k \in p(X)} G[k] \tag{20}$$

where G[k] *is the element at position* k *of* G.

Example 14. *For* DB_e, p({a}) = {1, 2, 3} *because* P_1, P_2, *and* P_3 *include* {a} *as a component. Similarly,* p({b}) = {1, 2, 4, 5, 10}*. According to Definition 12, the pidset of itemset* X = {ab} *is* p(X) = p({a}) ∪ p({b}) = {1, 2, 3} ∪ {1, 2, 4, 5, 10} = {1, 2, 3, 4, 5, 10}*. The gain of* X *is* g(X) = G[1] + G[2] + G[3] + G[4] + G[5] + G[10] = 4450 *dollars.*

Theorem 8. *Let* X *be a* k*-itemset and* B *be a* 1*-itemset. Assume that the pidset of* X *is* p(X) *and that of* B *is* p(B)*. Then:*

$$p(XB) = p(X) \quad p(B) \tag{21}$$

Theorem 9. *Let* XA *and* XB *be two itemsets with the same prefix* X*. Assume that* p(XA) *and* p(XB) *are pidsets of* XA *and* XB*, respectively. The pidset of* XAB *is computed as follows:*

$$p(XAB) = p(XB) \quad p(XA) \tag{22}$$

Example 15. *For* DB_e, XA = {ab} *with* p(XA) = {1, 2, 3, 4, 5, 10} *and* XB = {ac} *with* p(XB)= {1, 2, 3, 4, 6, 7, 11}*. According to Theorem 9, the pidset of itemset* XAB *is* p(XAB) = p(XBA) = p(XA) ∪ p(XB) = {1, 2, 3, 4, 5, 10} {1, 2, 3, 4, 6, 7, 11} = {1, 2, 3, 4, 5, 6, 7, 10, 11}.

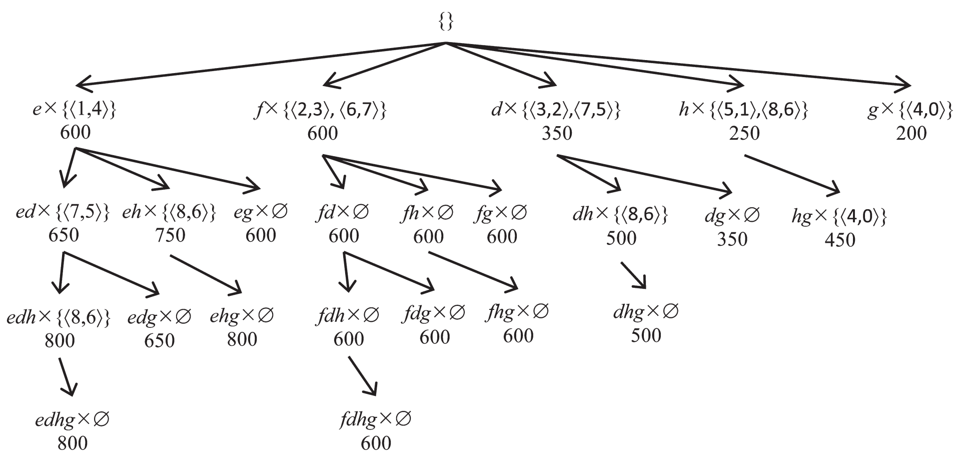

FIGURE 15 | Complete set of erasable itemsets identified by dMERIT+ for *DBe* with $\xi = 16\%$.

TABLE 13 | Index of Gain for DB_e

Index	1	2	3	4	5	6	7	8	9	10	11
Gain	2100	1000	1000	150	50	100	200	100	50	150	100

```
Procedure Sub_dPidsets
Input: dPidsets d1, d2 and index of gain G
Output: dPidset (d3) and its gain (Gain)
let i ← 0, j ← 0, Gain ← 0 and d3 ← ∅
while i < |d1| and j < |d2| do
   if d1[i] < d2[j] then
      i++
   else if d1[i] == d2[j] then
      i++ and j++
   else
      Gain = Gain + G[d2[j]]
      insert d2[j] into d3
      j++
while j < |d2| do
   Gain = Gain + G[d2[j]]
   insert d2[j] into d3
   j++
return d3 and Gain
```

FIGURE 16 | Efficient algorithm for subtracting two dPidsets.

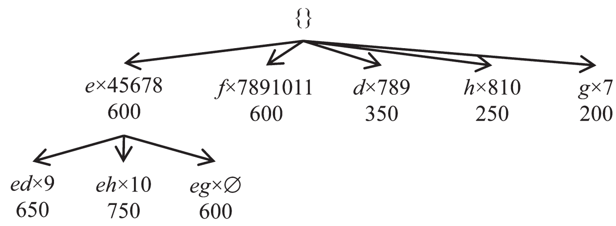

FIGURE 18 | Erasable 2-itemsets of node {e} for *DBe with* $\xi = 16\%$.

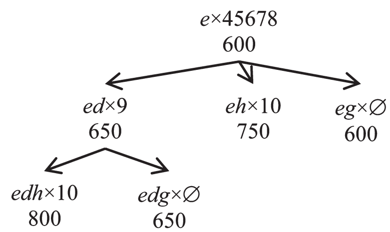

FIGURE 19 | Erasable 3-itemsets of node {*ed*} for *DBe* with $\xi = 16\%$.

dPidset—The Difference Pidset of Two Pidsets

Definition 14. *(dPidset) Let* XA *and* XB *be two itemsets with the same prefix* X. *The dPidset of pidsets* p(XA) *and* p(XB), *denoted as* dP(XAB), *is defined as follows:*

$$dP(XAB) = p(XB) \setminus p(XA) \tag{23}$$

According to Definition 14, the dPidset of pidsets p(XA) *and* p(XB) *is the product identifiers which only exist on* p(XB).

Example 16. XA = {ab} *with* p(XA) = {1, 2, 3, 4, 5, 10} *and* XB = {ac} *with* p(XB)= {1, 2, 3, 4, 6, 7, 11}. *Based on Definition 14, the dPidset of* XAB *is* dP(XAB) = p(XB) \ p(XA) = {1, 2, 3, 4, 6, 7, 11}\{1, 2, 3, 4, 5, 10} = {6, 7, 11}. *Note that reversing the order of* XA *and* XB *will get a different result.* dP(XBA) = p(XA) \ p(XB) = {5, 10}.

```
Input: product dataset DB and threshold ξ
Output: E_result, the set of all EIs
scan DB to determine the total profit of DB (T), the index of
gain (G), and erasable 1-itemsets with their pidsets (E1)
sort E1 by the length of their pidsets
E_result ← E1
if |E1| > 1 then
   call Expand_E(E1)

Procedure Expand_E(Ev)
for k ← 0 to Ev.size - 2 do
   E_next ← ∅
   for j ←(k+1) to |Ev| - 1 do
      E.Items = Ev[k].Items ∪ Ev[j].Items
      (E.pidset, Gain) ← Sub_dPidsets(Ev[k].pidset, Ev[j].pidset)
      E.gain = Ev[k].gain + Gain
      if E.gain < T × ξ then
         E_next ← E and E_result ← E
   if |E_next| > 1 then
      Expand_E(E_next)
```

FIGURE 17 | MEI algorithm.

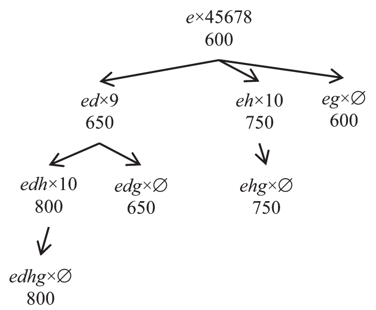

FIGURE 20 | All EIs of node {e} for *DBe* with $\xi = 16\%$.

Theorem 10. *Given an itemset* XY *with dPidset* dP*(*XY*) and pidset* p*(*XY*):*

$$dP\,(XY) \subset p\,(XY) \tag{24}$$

Example 17. *According to* Example 15, p*(*XAB*) =* p*(*{abc}*) = {1, 2, 3, 4, 5, 6, 7, 10, 11}. According to* Example 16, dP*(*XAB*) =* dP*(*{abc}*) = {6, 7, 11}. From this result,* dP*(*XAB*) = {6, 7, 11}* ⊂ p*(*XAB*) = {1, 2, 3, 4, 5, 6, 7, 8, 10, 11}. This example verifies Theorem 10.*

With an itemset *XY*, Theorem 10 shows that using dPidset is always better than using pidset because the algorithm will (1) use less memory and (2) require less mining time due to fewer elements.

Theorem 11. *Let* XA *and* XB *be two itemsets with the same prefix* X*. Assume that* dP*(*XA*) and* dP*(*XB*) are the dPidsets of* XA *and* XB*, respectively. The dPidset of* XAB *is computed as follows:*

$$dP\,(XAB) = dP\,(XB) \setminus dP\,(XA) \tag{25}$$

Example 18. *Let* X = {a}*,* A = {b}*, and* B = {c} *be three itemsets. Then,* p*(*X*) = {1, 2, 3},* p*(*A*) = {1, 2, 4, 5, 10}, and* p*(*B*) = {1, 3, 4, 6, 7, 11}.*

1. *According to Theorem 8,* p*(*XA*) =* p*(*X*) ∪* p*(*A*) = {1, 2, 3, 4, 5, 10} and* p*(*XB*) =* p*(*X*) ∪* p*(*B*) = {1, 2, 3, 4, 6, 7, 11}. Based on Definition 14, the dPidset of* XAB *is* dP*(*XAB*) =* p*(*XB*) \\ *p*(*XA*) = {1, 2, 3, 4, 6, 7, 11}\\{1, 2, 3, 4, 5, 10} = {6, 7, 11}.*
2. *According to Definition 14,* dP*(*XA*) =* p*(*A*) * p*(*X*) = {4, 5, 10} and* dP*(*XB*) =* p*(*B*) * p*(*X*) = {4, 6, 7, 11}. Based on Theorem 11, the dPidset of* XAB *is* dP*(*XAB*) =* dP*(*XB*) * dP*(*XA*) = {4, 6, 7, 11}\\{4, 5, 10} = {6, 7, 11}.*

In (1) and (2), the dPidset of XAB *is* dP*(*XAB*) = {6, 7, 11}. This example verifies Theorem 11.*

Theorem 12. *Let* XAB *be an itemset. The gain of* XAB *is determined based on that of* XA *as follows:*

$$g\,(XAB) = g\,(XA) + \sum_{P_k \in dP(XAB)} G\,[k] \tag{26}$$

where g(XA) *is the gain of* X *and* G[k] *is the element at position* k *of* G.

Example 19. *According to* Example 15, XA *=* {ab} *with* p*(*XA*) = {1, 2, 3, 4, 5, 10} and* XB *=* {ac} *with* p*(*XB*) = {1, 2, 3, 4, 6, 7, 11}. Applying Definition 13 yields* g*(*XA*) = 4,450 dollars and* g*(*XB*) = 4,650.*

1. *Based on Theorem 9,* p*(*XAB*) = {1, 2, 3, 4, 5, 6, 7, 10, 11}. Thus, the gain of* XAB *is* g*(*XAB*) = 4850.*
2. *According to Definition 14,* dP*(*XAB*) =* p*(*XB*) * p*(*XA*) = {1, 2, 3, 4, 6, 7, 11} \\{1, 2, 3, 4, 5, 10} = {6, 7, 11}. Therefore, the gain of* XAB *based on Theorem 12 is* $g\,(XAB) = g\,(XA) + \sum_{P_k \in d(XAB)} G\,[k]$ *= 4450 +* G*[6] +* G*[7] +* G*[11] = 4850.*

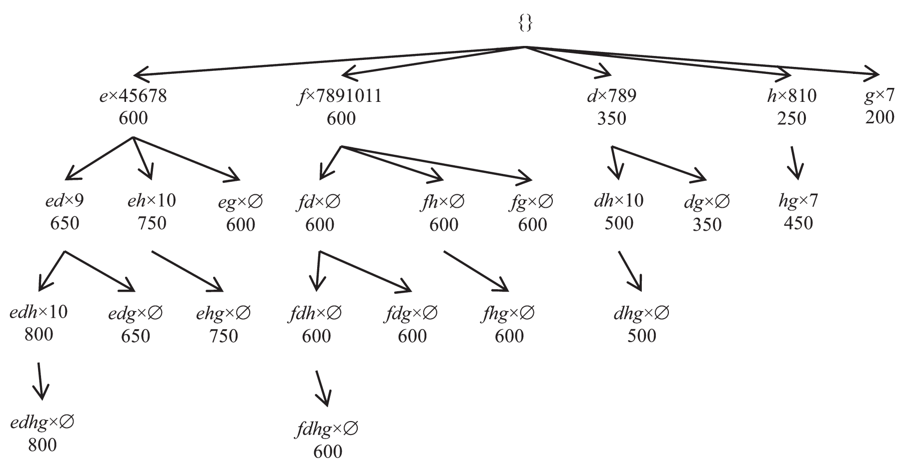

FIGURE 21 | Tree of all EIs obtained by MEI for *DBe* with $\xi = 16\%$.

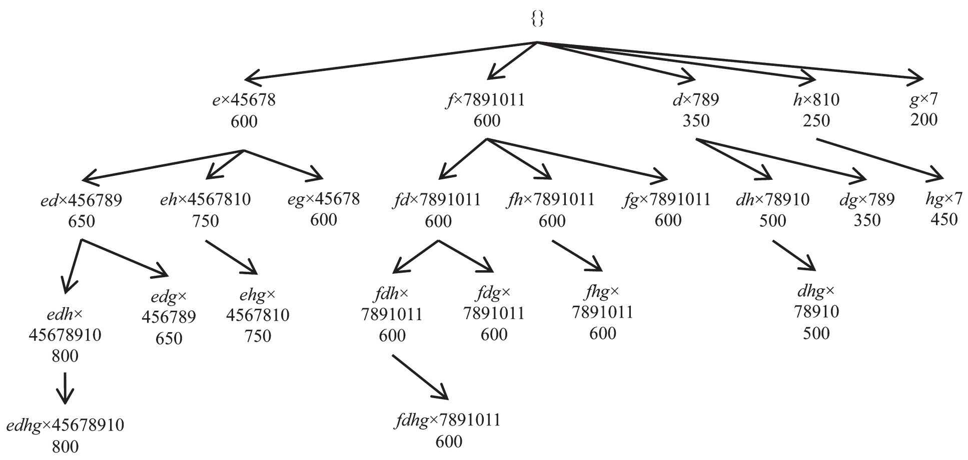

FIGURE 22 | Tree of EIs obtained using pidset for *DBe* with $\xi = 16\%$.

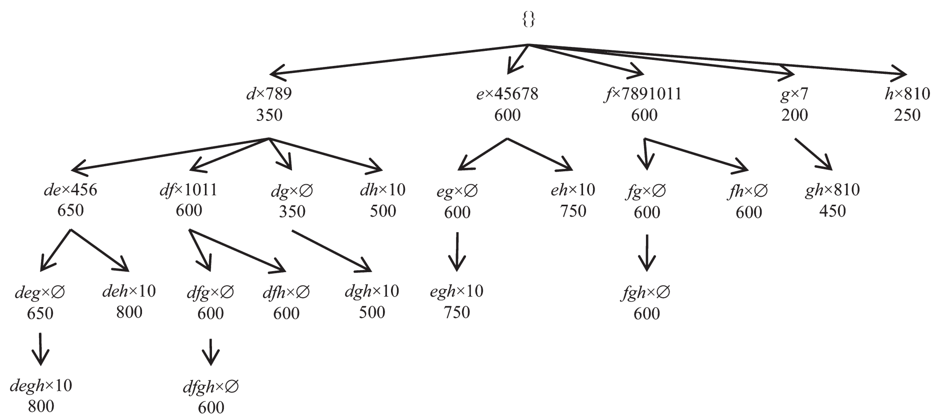

FIGURE 23 | Tree of EIs obtained using dPidset without sorting erasable 1-itemsets for *DBe* with $\xi = 16\%$.

TABLE 14 | Features of Synthetic Datasets Used in Experiments

Dataset[1]	No. Products	No. Items	Type of dataset
Accidents	340,183	468	Dense
Chess	3196	76	Dense
Connect	67,557	130	Dense
Mushroom	8124	120	Sparse
Pumsb	49,046	7,117	Dense
T10I4D100K	100,000	870	Sparse

[1]These databases are available at http://sdrv.ms/14eshVm

In (1) and (2), the gain of XAB is 4850 dollars. This example verifies Theorem 12.

Theorems 11 and 12 allow MEI to store the dPidset of erasable k-itemsets ($k \geq 2$) and easily determine the gain of erasable k-itemsets. MEI scans the dataset to create erasable 1-itemsets and their pidsets. Then, MEI combines all erasable 1-itemsets together to create erasable 2-itemsets and their dPidsets according to Definition 14. From erasable k-itemsets ($k \geq 2$), MEI uses Theorem 11 to determine their dPidsets and uses Theorem 12 to compute their gains.

Theorem 13. *Let* A *and* B *be two 1-itemsets. Assume that their pidsets are* p(A) *and* p(B)*, respectively. If* $|p(A)| > |p(B)|$*, then:*

$$|dP(AB)| < |dP(BA)| \tag{27}$$

Example 20. *Let* A = {a}, B = {b} *be two itemsets. Based on* DB_e, p(A) = *{1, 2, 3} and* p(B) = *{1, 2, 4, 5, 10}. Then:*

1. dP(AB) = p(B) \ p(A) = *{1, 2, 4, 5, 10}\{1, 2, 3} = {4, 5, 10}.*
2. dP(BA) = p(A) \ p(B) = *{1, 2, 3}\{1, 2, 4, 5, 10} = {3}.*

From (1) and (2), the size of dP(AB) *is larger than that of* dP(BA)*. This example verifies Theorem 13.*

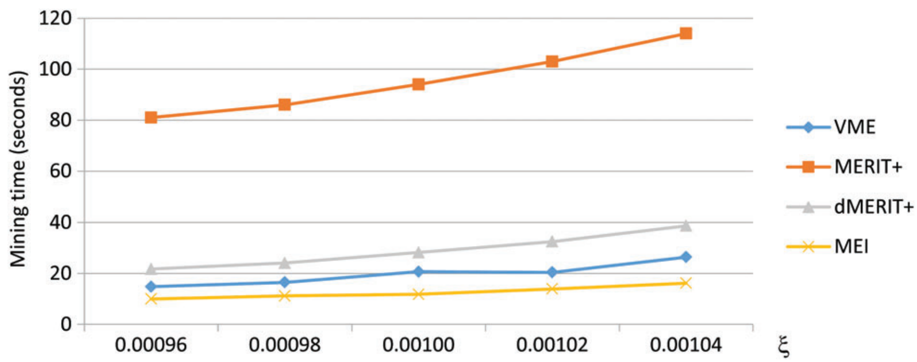

FIGURE 24 | Mining time for Accidents dataset.

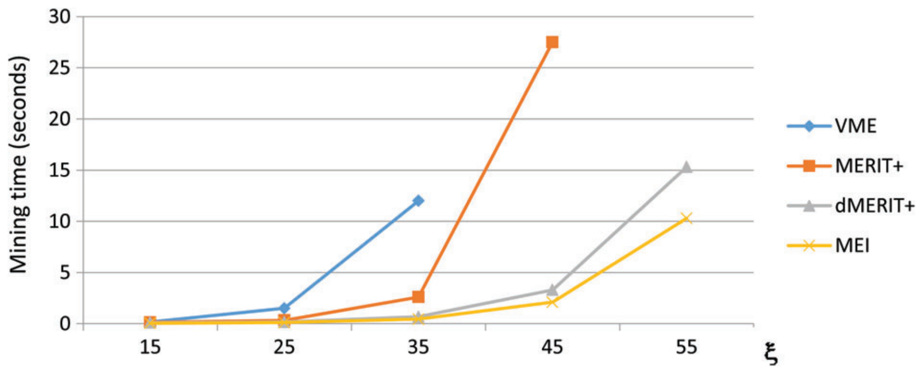

FIGURE 25 | Mining time for Chess dataset.

Theorem 13 shows that subtracting pidset d_2 from pidset d_1 with $|d_1| > |d_2|$ is always better in terms of memory usage and mining time compared to the reverse (subtracting pidset d_2 from pidset d_1 with $|d_1| < |d_2|$). Therefore, sorting erasable 1-itemsets in descending order of their pidset size before combining them together improves the algorithm. From the above analysis, MEI sorts erasable 1-itemsets in descending order of their pidset size.

Theorem 14. *Let* XA *and* XB *be two EIs with dPidsets* dP(XA) *and* dP(XB)*, respectively. If* $|dP(XA)| > |dP(XB)|$, *then:*

$$|dP(XAB)| < |dP(XBA)| \tag{28}$$

Example 21. *Let* XA = {ab} *with* dP(XA) = {4, 5, 10} *and* XB = {ac} *with* dP(XB) = {4, 6, 7, 11}. *Then:*

1. dP(XAB) = dP(XB) \ dP(XA) = {4, 6, 7, 11}\{4, 5, 10} = {6, 7, 11}.
2. dP(XBA) = dP(XA) \ dP(XB) = {4, 5, 10}\{4, 6, 7, 11} = {5, 10}.

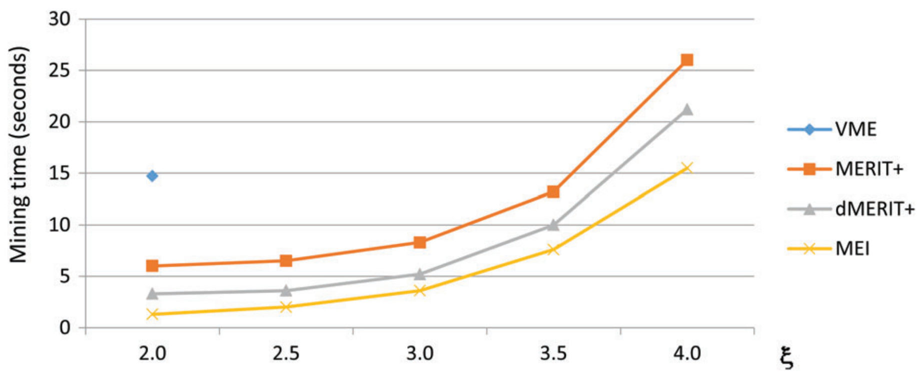

FIGURE 26 | Mining time for Connect dataset.

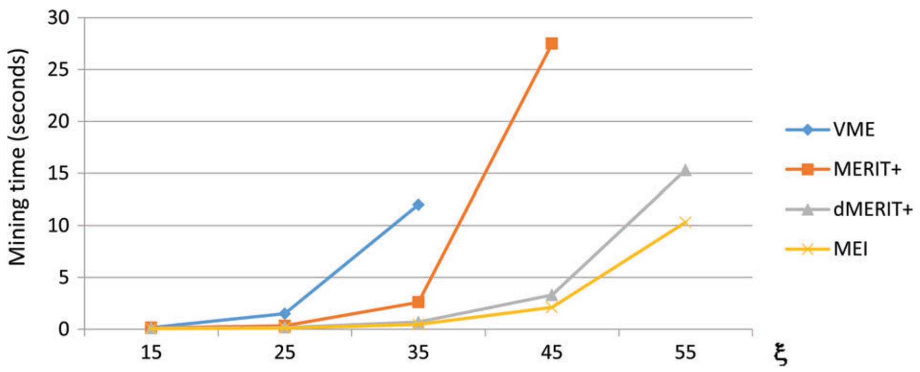

FIGURE 27 | Mining time for Mushroom dataset.

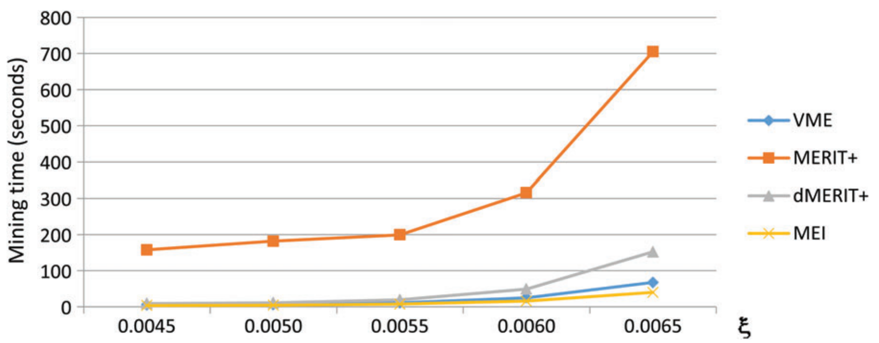

FIGURE 28 | Mining time for Pumsb dataset.

From (1) and (2), the size of dP(XBA) *is smaller than that of* dP(XAB)*. This example verifies Theorem 14.*

Theorem 14 shows that subtracting dPidset d_2 from dPidset d_1 with $|d_1| > |d_2|$ is always better in terms of memory usage compared to the reverse. Thus, sorting erasable k-itemsets ($k > 1$) in descending order of their dPidset size helps the algorithm optimize memory usage. However, Theorem 13 sorts erasable 1-itemsets in descending order of their pidset size. Hence, in most cases, the dPidsets of erasable k-itemsets ($k > 1$) are randomly sorted (see Section named "An illustrative example" for illustration). In these cases, this arrangement increases mining time. Therefore, MEI does not sort erasable k-itemsets ($k > 1$).

Effective Method for Subtracting Two dPidsets

In the conventional method, when subtracting dPidset d_2 with n elements from dPidset d_1 with m elements, the algorithm must consider every element in d_2 regardless of whether it exists in d_1. Therefore, the complexity of this method is $O(n \times m)$. After obtaining d_3 with k elements, the algorithm has to scan all elements in d_3 to determine the gain of an itemset.

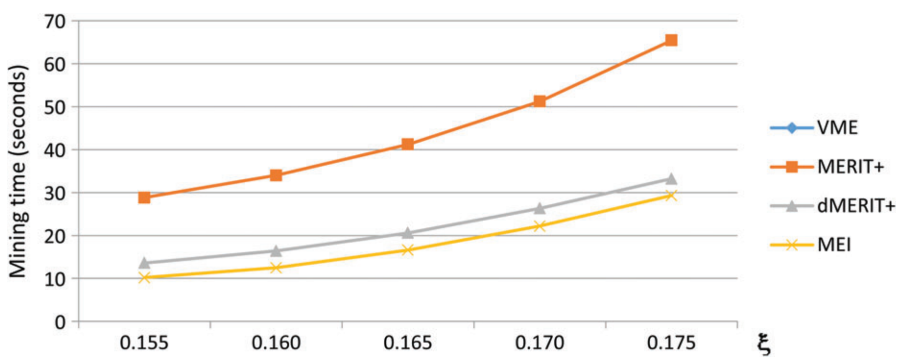

FIGURE 29 | Mining time for T10I4D100K dataset.

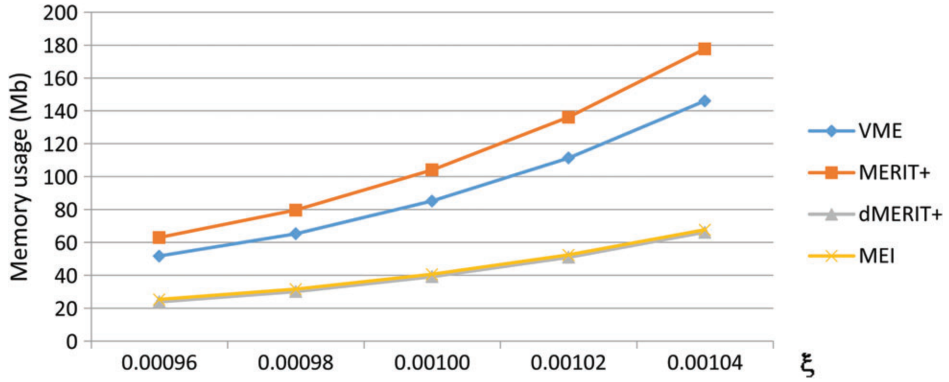

FIGURE 30 | Memory usage for Accidents dataset.

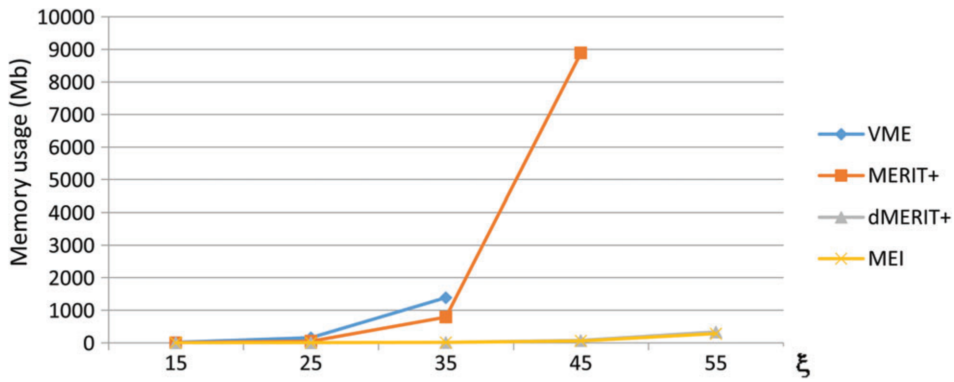

FIGURE 31 | Memory usage for Chess dataset.

The complexity of this algorithm is thus $O(n \times m + k)$. The mining time of the algorithm is not significant for the example dataset; however, it is very large for large datasets. Therefore, an effective method for subtracting two dPidsets is necessary.

In the process of scanning the dataset, MEI finds erasable 1-itemset pidsets, which are sorted in ascending order of the product identifiers. An effective algorithm for subtracting two dPidsets called *Sub_dPidsets* is proposed and shown in Figure 16.

Mining EIs Using dPidset Structure

The MEI algorithm[46] for mining EIs is shown in Figure 17. First, the algorithm scans the product dataset only one time to determine the total profit of the factory (T), the index of gain (G), and the erasable 1-itemsets with their pidsets. A divide-and-conquer strategy for mining EIs is proposed. First, with erasable k-itemsets, the algorithm combines the first element with the remaining elements in erasable k-itemsets to create the erasable $(k+1)$-itemsets. For elements whose gain is smaller than $T \times \xi$, the algorithm will (a) add them to the results of this algorithm and then (b) combine them together to create erasable $(k+2)$-itemsets. The algorithm uses this strategy until all itemsets which can be created from n elements of erasable 1-itemsets are considered.

An Illustrative Example

To demonstrate MEI, its implementation for DB_e with threshold $\xi = 16\%$ is described. The algorithm has the following four main steps:

1. MEI scans DB_e to determine T = 5000 dollars, the total profit of the factory, G, the index of gain, and the erasable 1-itemsets $\{d, e, f, g, h\}$ with their pidsets.
2. The erasable 1-itemsets are sorted in descending order of their pidset size. After sorting, the new order of erasable 1-itemsets is $\{e, f, d, h, g\}$.
3. MEI puts all elements in the erasable 1-itemsets into the results.
4. MEI uses the *Expand_E* procedure to implement the divide-and-conquer strategy. First, the first element of erasable 1-itemsets $\{e\}$ is combined in turn with the remaining elements of erasable 1-itemsets $\{f, d, h, g\}$ to create erasable 2-itemsets of node $\{e\}$: $\{ef, ed, eh, eg\}$. However,

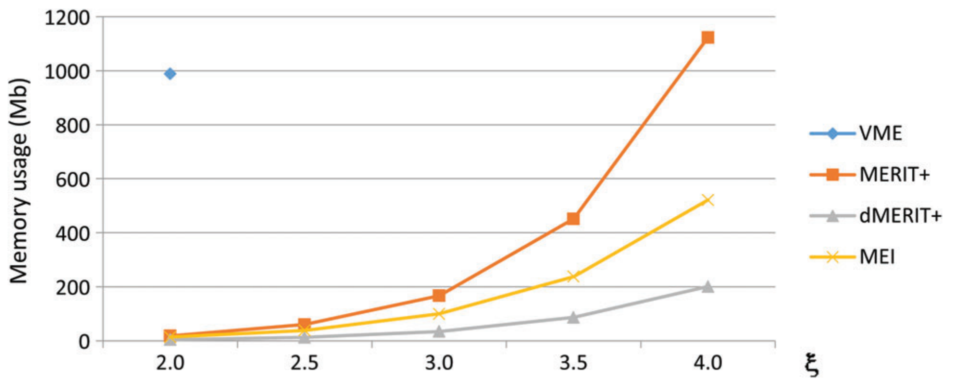

FIGURE 32 | Memory usage for Connect dataset.

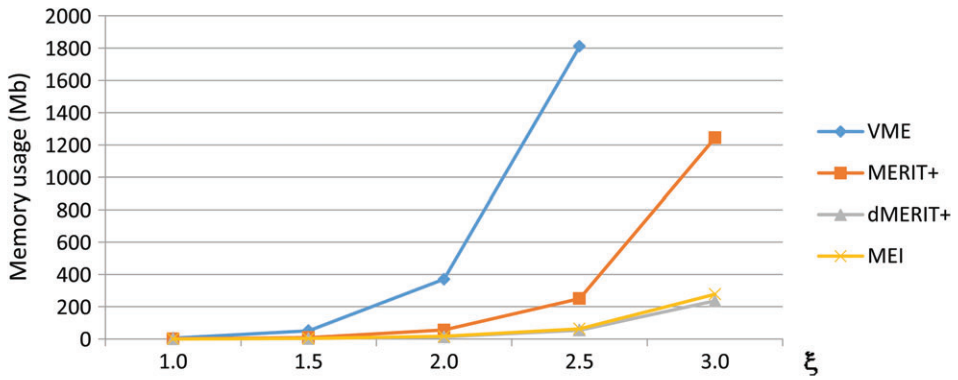

FIGURE 33 | Memory usage for Mushroom dataset.

{*ef*} is excluded because $g(ef) = 900 > T \times \xi$. Therefore, the erasable 2-itemsets of node {*e*} are {*ed*, *eh*, *eg*}, as illustrated in Figure 18.

The algorithm adds {*ed*, *eh*, *eg*}, the obtained erasable 2-itemsets of node {*e*}, to the results and uses them to call the *Expand_E* procedure to create erasable 3-itemsets. {*ed*}, the first element of the erasable 2-itemsets of node {*e*}, is combined in turn with the remaining elements {*eh*, *eg*}. The erasable 3-itemsets of node {*ed*} are {*edh*, *edg*} because their gain is less than $T \times \xi$. The erasable 3-itemsets of node {*ed*} are illustrated in Figure 19.

The algorithm is called recursively in depth-first order until all EIs of node {*e*} are created. Figure 20 shows all EIs of node {*e*}.

Then, the algorithm continues to combine the next element {*f*} with the remaining elements of erasable 1-itemsets {*d*, *h*, *g*} to create the EIs of node {*f*}. The algorithm repeats until all nodes are considered. Then, the algorithm obtains the tree of all EIs, as shown in Figure 21.

The memory usage for pidset and dPidset is compared to show the effectiveness of using dPidset. The EI tree obtained using the pidset strategy for DB_e for $\xi = 16\%$ is shown in Figure 22.

According to Figure 22, using pidset leads to data duplication. Assume that each product identifier is represented as an integer (4 bytes in memory). The size of pidsets in Figure 22 is $106 \times 4 = 424$ bytes. The algorithm with dPidset (Figure 21) only uses $21 \times 4 = 84$ bytes. Therefore, the memory usage with pidset is larger than that with dPidset.

The memory usage of the algorithm using dPidset with (see Figure 21) and without sorting pidsets was determined. The tree of the EIs obtained using dPidset without sorting for DB_e with $\xi = 16\%$ is shown in Figure 23.

The algorithm is better with sorting erasable 1-itemsets than it is without, as discussed in Theorem 13. The algorithm using dPidset with sorting erasable 1-itemsets requires $21 \times 4 = 84$ bytes (see Figure 21) whereas that without sorting erasable 1-itemsets requires $29 \times 4 = 116$ bytes (see Figure 23). This difference in memory usage is significant for real datasets. In addition, reducing the memory usage also speeds up the algorithm. Therefore, the algorithm with sorting erasable 1-itemsets is better than that without sorting erasable 1-itemsets.

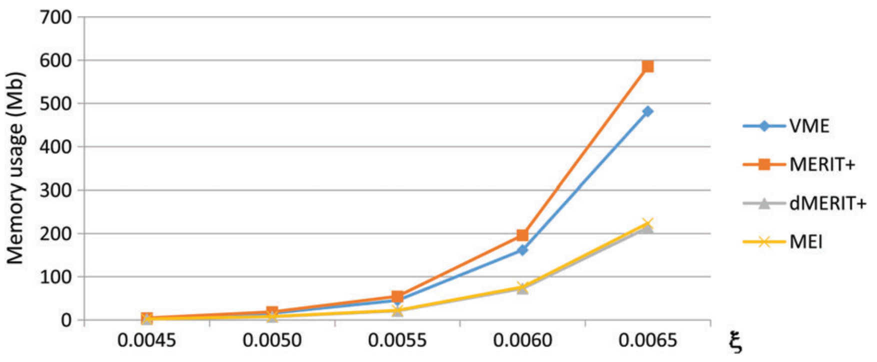

FIGURE 34 | Memory usage for Pumsb dataset.

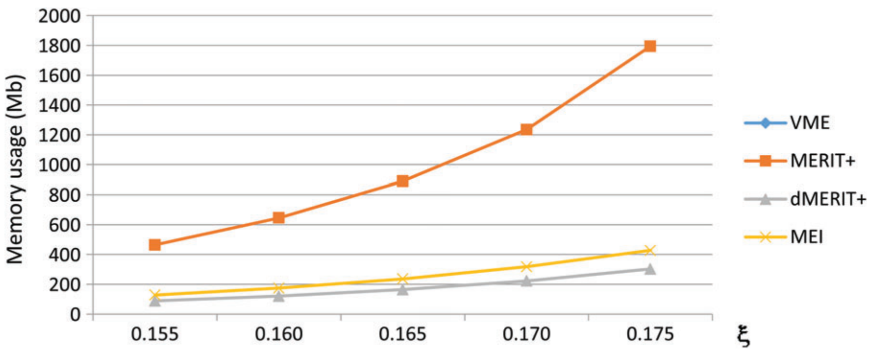

FIGURE 35 | Memory usage for T10I4D100K dataset.

EXPERIMENTAL RESULTS

Experimental Environment

All experiments presented in this section were performed on a laptop with an Intel Core i3-3110 M 2.4-GHz CPU and 4 GB of RAM. The operating system was Microsoft Windows 8. All the programs were coded in C# using Microsoft Visual Studio 2012 and run on Microsoft .Net Framework Version 4.5.50709.

Experimental Datasets

The experiments were conducted on synthetic datasets Accidents, Chess, Connect, Mushroom, Pumsb, and T10I4D100K.[a] To make these datasets look like product datasets, a column was added to store the profit of products. To generate values for this column, a function denoted by $N(100,50)$ was created. For each product, this function returned a value, with the mean and variance of all values being 100 and 50, respectively. In other words, this function created a random value r $(-50 \leq r \leq 50)$, and returned $100 + r$ for this value. The features of these datasets are shown in Table 14.

Discussion

Because of the loss of EIs with MERIT, it is unfair to compare MERIT to algorithms which mine all EIs (VME, dMERIT+, and MEI) in terms of mining time and memory usage. Therefore, MERIT+, derived from MERIT (see Section 4.4), was implemented for mining all EIs. The mining time and memory usage of VME, MERIT+, dMERIT+, and MEI were compared. Note that:

1. The mining time is the total execution time; i.e., the period between input and output.
2. The memory usage is determined by summing the memory which stores: (1) EIs and their dPidset (MEI), or (2) EIs, their NC_Set, and the WPPC-tree (MERIT+), or (3) EIs, their dNC'_Set, and the WPPC-tree (dMERIT+), or (4) EIs and their PID_Lists (VME).

Mining Time

The mining time of MEI is always smaller than those of VME and MERIT+ (Figures 24–29). This can be explained primarily by the union PID_List strategy of VME, the union NC_Set strategy of MERIT+, the dNC_Set strategy of dMERIT+, and the dPidset strategy of MEI. The union PID_List and NC_Set strategies require a lot of memory and many operations, making the mining times of VME and MERIT+ long. The dNC_Set and dPidset strategies reduce the number of operations and thus the mining time.

For Accidents, Pumsb, and T10I4D100K datasets, MERIT+ and dMERIT+ take a lot of time to build the WPPC-tree. Therefore, the mining times of MERIT+ and dMERIT+ are much larger than that of MEI (see Figures 24, 28 and 29) for datasets with a large number of items. For Chess, Connect, Mushroom, and T10I4D100K, VME cannot run with some thresholds (Figures 25–27 and 29). It can only run with threshold = 2% for Connect (Figure 26) and cannot run with $0.155\% \leq$ threshold $\leq 0.175\%$ for T10I4D100K (Figure 29).

Memory Usage

VME and MERIT+ use the union strategy whereas dMERIT+ and MEI use the difference strategy. The memory usage associated with dMERIT+ and MEI is much smaller than that of VME and MERIT+ (see Figures 30–35). Because dMERIT+ and MEI reduce memory usage, they can mine EIs with thresholds higher than those possible for VME and MERIT+ for datasets such as Chess (see Figure 31). dMERIT+ and MEI can run with $45\% < \xi \leq 55\%$ for Chess but VME and MERIT+ cannot. In addition, VME cannot run for some datasets (Figures 31–33 and 35) with high thresholds.

From Figures 30–35, dMERIT+ and MEI have the same memory usage. They both outperform VME and MERIT+ in terms of memory usage and can mine EIs with higher thresholds.

Discussions and Analysis

According to the experimental results in Section 4.5, dMERIT+ uses slightly less memory that does MEI. Especially for Connect, dMERIT+ is much better than MEI in terms of memory usage. However, the memory usage difference between dMERIT+ and MEI is not significant for most of the tested datasets (Accidents, Chess, Pumsb, and T10I4D100K). dMERIT+'s mining time is always longer than MEI's mining time. Therefore, MEI is the best algorithm for mining EIs in terms of the mining time for all datasets. Finally, it can be concluded that META < VME < MERIT+ < dMERIT+ < MEI, where A < B means that B is better than A in terms of the mining time and memory usage. However, for cases with limited memory, users should consider using dMERIT+ instead of MEI.

CONCLUSIONS AND FUTURE WORK

This article reviewed the META, VME, MERIT, dMERIT+, and MEI algorithms for mining EIs. The theory behind each algorithm was described and weaknesses were discussed. The approaches were compared in terms of mining time and memory usage. Based on the results, MEI is the best algorithm for mining EIs. However, for cases with limited memory, dMERIT+ should be used.

In future work, some issues related to EIs should be studied, such as mining EIs from huge datasets, mining top-rank-*k* EIs, mining closed/maximal EIs, and mining EIs from incremental datasets. In addition, how to mine rules from EIs and how to use EIs in recommendation systems should be studied.

NOTE

[a] Downloaded from http://fimi.cs.helsinki.fi/data/

REFERENCES

1. Agrawal R, Srikant R. Fast algorithms for mining association rules. In: *Proceedings of the International Conference on Very Large Databases*, Santiago de Chile, Chile, 1994, 487–499.
2. Zaki MJ, Parthasarathy S, Ogihara M, Li W. New algorithms for fast discovery of association rules. In: *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, Newport Beach, California, USA, 1997, 283–286.
3. Lin KC, Liao IE, Chen ZS. An improved frequent pattern growth method for mining association rules. *Expert Syst Appl* 2011, 38:5154–5161.
4. Vo B, Le B. Interestingness measures for mining association rules: combination between lattice and hash tables. *Expert Syst Appl* 2011, 38:11630–11640.
5. Vo B, Hong TP, Le B. DBV-Miner: a dynamic bit-vector approach for fast mining frequent closed itemsets. *Expert Syst Appl* 2012, 39:7196–7206.
6. Vo B, Hong TP, Le B. A lattice-based approach for mining most generalization association rules. *Knowl-Based Syst* 2013, 45:20–30.
7. Abdi MJ, Giveki D. Automatic detection of erythemato-squamous diseases using PSO-SVM based on association rules. *Eng Appl Artif Intel* 2013, 26:603–608.
8. Kang KJ, Ka B, Kim SJ. A service scenario generation scheme based on association rule mining for elderly surveillance system in a smart home environment. *Eng Appl Artif Intel* 2012, 25:1355–1364.

9. Verykios VS. Association rule hiding methods. *WIREs Data Min Knowl Discov* 2013, 3:28–36.
10. Agrawal R, Gehrke J, Gunopulos D, Raghavan P. Automatic subspace clustering of high dimensional data for data mining applications. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Seattle, WA, 1998, 94–105.
11. Lin CW, Hong TP, Lu WH. The Pre-FUFP algorithm for incremental mining. *Expert Syst Appl* 2009, 36:9498–9505.
12. Nguyen LTT, Vo B, Hong TP, Thanh HC. Classification based on association rules: a lattice-based approach. *Expert Syst Appl* 2012, 39:11357–11366.
13. Nguyen LTT, Vo B, Hong TP, Thanh HC. CAR-Miner: an efficient algorithm for mining class-association rules. *Expert Syst Appl* 2013, 40:2305–2311.
14. Borgelt C. Frequent item set mining. *WIREs Data Min Knowl Discov* 2012, 2:437–456.
15. Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation. In: *International Proceedings of the 2000 ACM SIGMOD*, Dallas, TX, 2000, 1–12.
16. Zaki M, Gouda K. Fast vertical mining using diffsets. In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Washington, DC, USA, 2003, 326–335.
17. Vo B, Le T, Coenen F, Hong TP. A hybrid approach for mining frequent itemsets. In: *Proceedings of the 2013 IEEE International Conference on Systems, Man, and Cybernetics*, Manchester, UK, 2013, 4647–4651.
18. Hong TP, Lin CW, Wu YL. Maintenance of fast updated frequent pattern trees for record deletion. *Comput Stat Data Anal* 2009, 53:2485–2499.
19. Hong TP, Lin CW, Wu YL. Incrementally fast updated frequent pattern trees. *Expert Syst Appl* 2008, 34:2424–2435.
20. Lin CW, Hong TP, Lu WH. Using the structure of prelarge trees to incrementally mine frequent itemsets. *New Generat Comput* 2010, 28:5–20.
21. Lin CW, Hong TP. Maintenance of prelarge trees for data mining with modified records. *Inform Sci* 2014, 278:88–103.
22. Nath B, Bhattacharyya DK, Ghosh A. Incremental association rule mining: a survey. *WIREs Data Min Knowl Discov* 2013, 3:157–169.
23. Vo B, Le T, Hong TP, Le B. Maintenance of a frequent-itemset lattice based on pre-large concept. In: *Proceedings of the Fifth International Conference on Knowledge and Systems Engineering*, Ha Noi, Vietnam, 2013, 295–305.
24. Vo B, Le T, Hong TP, Le B. An effective approach for maintenance of pre-large-based frequent-itemset lattice in incremental mining. *Appl Intell* 2014, 41:759–775.
25. Lucchese B, Orlando S, Perego R. Fast and memory efficient mining of frequent closed itemsets. *IEEE Trans Knowl Data Eng* 2006, 18:21–36.
26. Zaki MJ, Hsiao CJ. Efficient algorithms for mining closed itemsets andtheir lattice structure. *IEEE Trans Knowl Data Eng* 2005, 17:462–478.
27. Hu J, Mojsilovic A. High-utility pattern mining: a method for discovery of high-utility item sets. *Pattern Recogn* 2007, 40:3317–3324.
28. Lin CW, Hong TP, Lu WH. An effective tree structure for mining high utility itemsets. *Expert Syst Appl* 2011, 38:7419–7424.
29. Lin CW, Lan GC, Hong TP. An incremental mining algorithm for high utility itemsets. *Expert Syst Appl* 2012, 39:7173–7180.
30. Liu J, Wang K, Fung BCM. Direct discovery of high utility itemsets without candidate generation. In: *Proceedings of the IEEE 12th International Conference on Data Mining*, Brussels, Belgium, 2012, 984–989.
31. Fan W, Zhang K, Cheng H, Gao J, Yan X, Han J, Yu P, Verscheure O. Direct mining of discriminative and essential frequent patterns via model-based search tree. In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Las Vegas, Nevada, USA, 2008, 230–238.
32. Gupta R, Fang G, Field B, Steinbach M, Kumar V. Quantitative evaluation of approximate frequent pattern mining algorithms. In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Las Vegas, Nevada, USA, 2008, 301–309.
33. Jin R, Xiang Y, Liu L. Cartesian contour: a concise representation for a collection of frequent sets. In: *Proceedings of ACM SIGKDD Conference*, Paris, 2009, 417–425.
34. Poernomo A, Gopalkrishnan V. Towards efficient mining of proportional fault-tolerant frequent itemsets. In: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Paris, 2009, 697–705.
35. Aggarwal CC, Li Y, Wang J, Wang J. Frequent pattern mining with uncertain data. In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data* Mining, Paris, 2009, 29–38.
36. Aggarwal CC, Yu PS. A survey of uncertain data algorithms and applications. *IEEE Trans Knowl Data Eng* 2009, 21:609–623.
37. Leung CKS. Mining uncertain data. *WIREs Data Min Knowl Discov* 2011, 1:316–329.
38. Lin CW, Hong TP. A new mining approach for uncertain databases using CUFP trees. *Expert Syst Appl* 2012, 39:4084–4093.
39. Wang L, Cheung DWL, Cheng R, Lee SD, Yang XS. Efficient mining of frequent item sets on large uncertain databases. *IEEE Trans Knowl Data Eng* 2012, 24:2170–2183.
40. Yun U, Shin H, Ryu KH, Yoon E. An efficient mining algorithm for maximal weighted frequent patterns

in transactional databases. *Knowl-Based Syst* 2012, 33:53–64.

41. Vo B, Coenen F, Le B. A new method for mining Frequent Weighted Itemsets based on WIT-trees. *Expert Syst Appl* 2013, 40:1256–1264.
42. Deng ZH. Mining top-rank-k erasable itemsets by PID_lists. *Int J Intell Syst* 2013, 28:366–379.
43. Deng ZH, Xu XR. Fast mining erasable itemsets using NC_sets. *Expert Syst Appl* 2012, 39:4453–4463.
44. Deng ZH, Fang G, Wang Z, Xu X. Mining erasable itemsets. In: *Proceedings of the 8th IEEE International Conference on Machine Learning and Cybernetics*, Baoding, Hebei, China, 2009, 67–73.
45. Deng ZH, Xu XR. An efficient algorithm for mining erasable itemsets. In: *Proceedings of the 2010 International Conference on Advanced Data Mining and Applications (ADMA)*, Chongqing, China, 2010, 214–225.
46. Le T, Vo B. MEI: an efficient algorithm for mining erasable itemsets. *Eng Appl Artif Intel* 2014, 27:155–166.
47. Le T, Vo B, Coenen F. An efficient algorithm for mining erasable itemsets using the difference of NC-Sets. In: *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, Manchester, UK, 2013, 2270–2274.
48. Nguyen G, Le T, Vo B, Le B. A new approach for mining top-rank-k erasable itemsets. In: *Sixth Asian Conference on Intelligent Information and Database Systems*, Bangkok, Thailand, 2014, 73–82.
49. Agrawal R, Imielinski T, Swami AN. Mining association rules between sets of items in large databases. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Washington DC, May 1993, 207–216.
50. Dong J, Han M. BitTableFI: an efficient mining frequent itemsets algorithm. *Knowl-Based Syst* 2007, 20:329–335.
51. Grahne G, Zhu J. Fast algorithms for frequent itemset mining using FP-trees. *IEEE Trans Knowl Data Eng* 2005, 17:1347–1362.
52. Song W, Yang B, Xu Z. Index-BitTableFI: an improved algorithm for mining frequent itemsets. *Knowl-Based Syst* 2008, 21:507–513.
53. Deng ZH, Wang ZH, Jiang JJ. A new algorithm for fast mining frequent itemsets using N-lists. *Sci China Inf Sci* 2012, 55:2008–2030.
54. Deng ZH, Wang Z. A new fast vertical method for mining frequent patterns. *Int J Comput Int Syst* 2010, 3:733–744.
55. Liu B, Hsu W, Ma Y. Integrating classification and association rule mining. In: *ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD'98)*, New York, NY, 1998, 80–86.

