International Journal of Database Theory and Application
Vol.9, No.5 (2016), pp.251-266
http://dx.doi.org/10.14257/ijdta.2016.9.5.26

An Improved Eclat Algorithm for Mining Association Rules Based on Increased Search Strategy

Zhiyong Ma, Juncheng Yang, Taixia Zhang and Fan Liu

Faculty of Mechanical Engineering and Mechanics,
Zhejiang Provincial Key Lab of Part Rolling Technology, Ningbo University
818 Fenghua Rd., Ningbo City, Zhejiang Province 315211, P. R. China
mazhiyong@nbu.edu.cn

Abstract

Although Eclat algorithm is an efficient algorithm for mining association rules, there are some disadvantages which limit the efficient of Eclat. In this paper, we proposed an improved Eclat algorithm called Eclat_growth which is based on the increased search strategy. There are three main steps in the Eclat_growth algorithm. First, it scans the database and stores it into a table using vertical data format. Then, it builds an increased two-dimensional pattern tree and the TID_sets of itemsets in the vertical data format table are added into the pattern tree row by row. New frequent itemsets are generated by combining the new added item data with the existing frequent itemsets in the pattern tree. Finally, all frequent itemsets can be found by picking up all nodes of the pattern tree. In the process of generating new frequent itemsets, the prior knowledge is used to fully clip the candidate itemsets. In the process of generating an intersection of two itemsets and calculating the support degree, we proposed a new method called BSRI (Boolean array setting and retrieval by indexes of transactions) to reduce the run time. By comparing Eclat_growth with Eclat, Eclat-diffsets, Eclat-opt and hEclat, it is indicated that Eclat_growth has the highest performance in mining associating rules from various databases.

Keywords*: association rules, Eclat, increased search strategy, increased two-dimensional pattern tree, BSRI*

1. Introduction

Along with the comprehensive application of computer technology, big data gets more and more attentions. There is much information hiding in big data. By using the technology of data mining, we can mine interesting information from vast data. Mining association rules is very important in data mining and can be used for mining frequent patterns from databases.

The basic concepts of association rule can be stated as follows: Let D be a database of transactions; T be a transaction, $T \subseteq D$. Let I be a set of items, the number of items in I is called the length of I. Let X be a subset of I ($X \subseteq I$) and X contains k items, we call X a k-itemset. Each transaction T has a unique identifier (TID) and contains an itemset. The support of an itemset X, denoted as Support(X), is the number of transactions which contain the itemset X. Set a minimum of support as min_sup, if Support(X) $\geq$ min_sup, the itemset X is a frequency itemset.

An association rule is an expression $A \Rightarrow B$, which represents the probability of item A and B arising together in the dataset D, where $A \subset I$, $B \subset I$, $A \neq \emptyset$, $B \neq \emptyset$, and $A \cap B = \emptyset$.

Two of the most classical algorithms for mining association rules are Apriori algorithm, proposed by R. Agrawal in 1993 [1], and FP-growth algorithm, proposed by Han in 2000 [2]. Apriori algorithm is based on the strategy of breadth-first search, which generates a ($K+1$)-itemsets form frequent k-itemsets, until no more frequent itemsets can

ISSN: 2005-4270 IJDTA

be found. Apriori needs to scan the database many times, its efficiency is limited by the number of candidate itemsets. According to the shortcoming of Apriori, many people proposed their improved algorithms[3-6], such as DHP, proposed by Pork et al. [7], DIC, proposed by Brin et al. [8], MFI-TransSW, proposed by H. F. Li et al. [9], and so on. Different from Apriori, FP-growth algorithm is based on the strategy of depth-first search, it does not need to generate candidate itemsets; instead, it compresses datasets into a FP-tree and obtains frequent patterns using an FP-tree-based pattern fragment growth mining method. On the base of FP-growth, many improved algorithms was proposed[10-12], such as TD-FP-Growth, proposed by Wang Ke et al. [13], H-Mine, proposed by Jian Pei et al. [14], IFP-growth, proposed by Ke-Chung Lin et al. [15], and so on. Above algorithms are all based on the horizontal data format. There are many other algorithms based on the vertical data format, such as Eclat and the proposed algorithm. Compared with horizontal data format, the technology of vertical data format can improve the efficiency of the algorithm. We will introduce the vertical data format and Eclat algorithm in the following.

The organization of this paper can be described as follows. In Section 2, we review the original Eclat algorithm and several improved Eclat algorithms. In Section 3, we give the Eclat_growth algorithm. In Section 4, we present the experimental results, and conclusions are given in Section 5 finally.

2. Eclat and Improved Algorithms

2.1. Data Format of Eclat

There are two different data formats: horizontal data format and vertical data format. The data format of Eclat is vertical data format. Both data formats are the representations of database in memory. Horizontal data format, a "TID-itemset" format, can also be expressed as "TID: itemset", in which "TID" is the unique identifier of a transaction in database, "itemset" is the set of items included in the transaction. Vertical data format, an "Item-TID_set" format, can also be expressed as "Item: TID_set", in which "Item" is an item in database, "TID_set" is the set of transactions including the items. Table 1 shows the horizontal data format representation of a database and Table 2 shows the corresponding vertical data format representation. Two data formats have different efficiency in calculating the support degree of itemset. When adopting the horizontal data format, it needs to scan the whole database to get the support degree of itemset. But if adopting the vertical data format, it only needs to calculate the intersection of all TID_sets of items to get the support degree. For example, when we want to get the support of an itemset X={*I1*,*I2*}, for the data in Table 1, we need to scan all transactions from *TID 1* to *TID 6* to get all transactions including item *I1* and *I2* (1、3、5 and 6), then Support(*I1*,*I2*)=4. For the data in Table 2, we only need to calculate the intersection of TID_sets of *I1* and *I2* and the number of TIDs in the intersection, so Support(*I1*,*I2*)= Num((1,3,4,5,6)∩(1,3,5,6))=Num(1,3,5,6)=4. This example shows that adopting the vertical data format can reduce the time of scanning database effectively and improve the efficiency of getting support degree.

Table 1. Horizontal Data Format of a Database

TID	Itemsets
1	I0,I1,I2,I4
2	I0,I3,I5,I6
3	I1,I2,I3,I4,I6
4	I1,I4,I5,I6
5	I0,I1,I2,I5,I6
6	I1,I2,I5,I6

Table 2. Vertical Data Format of a Database

Item	TID_set
I0	1,2,5
I1	1,3,4,5,6
I2	1,3,5,6
I3	2,3
I4	1,3,4
I5	2,4,5,6
I6	2,3,4,5,6

2.2. Eclat Algorithm

Eclat algorithm, proposed by ZAKI in 2000, is based on the breadth-first search strategy, which adopts the technologies of vertical data format, lattice theory, equivalence classes, intersection and so on [16]. The main steps of Eclat are listed as follows: scan the database to get all frequent 1-itemsets, generate candidate 2-itemsets from frequent 1-itemsets, then get all frequent 2-itemsets by clipping non-frequent candidate itemsets; generate candidate 3-itemsets from frequent 2-itemsets and then get all frequent 3-itemsets by clipping non-frequent candidate itemsets; repeat the above steps, until no candidate itemset can be generated.

Same as Apriori, Eclat algorithm also adopts the join operation ($\bowtie$) to generate candidate ($K+1$)-itemset by taking the union of two k-itemset. The condition of two k-itemset can be joined is that the front $k-1$ items of the two k-itemset must be the same. For example, there are two 3-itemset: $l_{31}=\{I1,I2,I3\}$ and $l_{32}=\{I1,I2,I4\}$, the first and second items of l_{31} and l_{32} are the same, so l_{31} and l_{32} can be joined to generate a 4-itemset: $l_4=l_{31}\bowtie l_{32}=\{I1,I2,I3,I4\}$. By using the concept of equivalence classes, Eclat divides the search space into multiple nonoverlapping sub spaces. The itemsets which have same prefix can be classified into a same class, and the generation of candidate itemsets can be only operated in a same class. The technology of equivalence classes can obviously improve the efficiency of generating candidate itemset and can reduce the occupation of memory.

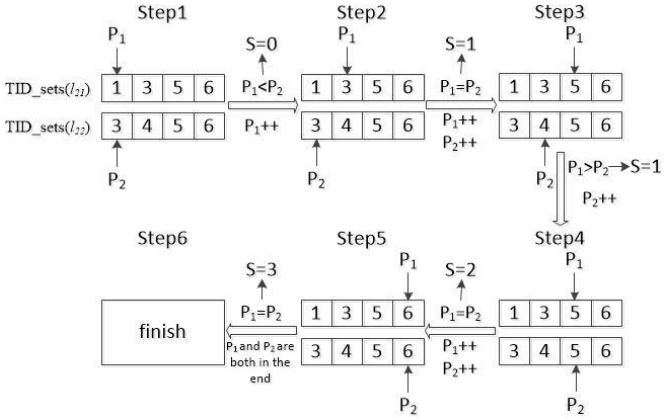


Figure 1. The Process of Getting Intersection

Let l_{21} and l_{22} are two itemset, TID_set(l_{21})={*1,3,5,6*}, TID_set(l_{22})={*3,4,5,6*}. Let $l_3 = l_{21} \bowtie l_{22}$, *Support*($l_3$)=*S*, TID_set($l_3$)= TID_set($l_{21}$) ∩TID_set($l_{22}$). Let P_1 and P_2 are two pointers respectively directing to the first TID of TID_set(l_{21}) and TID_set(l_{22}). The process of getting the intersection of l_{21} and l_{22} is shown in Figure 1. During the generation of a (*K+1*)-itemset, the intersection of the two *k*-itemset is calculated at the same time, so the support degree of the (*K+1*)-itemset can be obtained immediately.

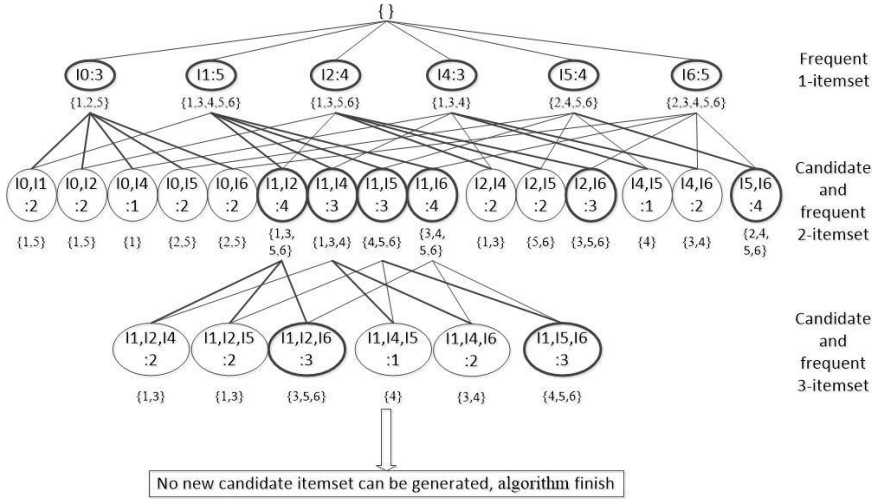


Figure 2. The Mining Process of Eclat

Taking the database shown in Table 2 as an example, let *min_sup*=3, the process of mining association rules for Eclat is demonstrated in Figure 2. In Figure 2, the data in ellipses are itemsets and its support degree and the data under an ellipse are the TID_sets of itemsets. The itemsets in bold ellipses are frequent itemsets, and the itemsets in light ellipses are candidate non-frequent itemsets. Figure 2 indicates that for the database in Table 2, 21 frequent itemsets are generated (eliminating the frequent 1-itemsets), 21 join and intersection operations are processed and 8 frequent itemsets are generated.

Although Eclat algorithm has a high efficiency, it still has some shortcomings as following. 1) Because the generation of candidate itemsets is operated in an equivalence class, the candidate itemsets are not clipped under the prior knowledge and the number of candidate itemsets is more than that of Apriori. Taking the mining process shown in Figure 2 as example, during the process of generation of candidate 3-itemsets from frequent 2-itemsets, four candidate 3-itemsets ({I_1,I_2,I_4},

$\{I_1,I_2,I_5\}$, $\{I_1,I_4,I_5\}$, $\{I_1,I_5,I_6\}$) should be clipped under the prior knowledge, however, in Eclat, they still need to be calculated. 2) Although adopting the technology of equivalence classes, Eclat needs to judge whether two k-itemsets can be joined to generate a $(k+1)$-itemsets, a great deal of time is needed if the itemset is very long. 3) The efficiency for calculating the intersection of itemsets is low, especially when the itemsets have a large number of transactions. Although Eclat has above shortcomings, it is still very effective [17]. Many people paid attention to Eclat and proposed some improved algorithm. We will introduce some improved Eclat algorithm in the following.

2.3. Existing Improved Eclat Algorithms

Eclat_Diffsets algorithm, proposed by ZAKI et al in 2001 [18], adopts Boolean matrix and Diffsets to enhance the efficiency of intersection and reduce the memory occupation. The algorithm adopts Boolean matrix to store the itemset and TID_set, and uses binary operation to calculate the intersection, which can obviously improve the efficiency of intersection. Diffsets only keeps track of differences in the TID_sets of a candidate itemset from its generating frequent itemsets, it can effectually reduce the size of memory required to store intermediate results when the database is compact.

HEclat algorithm, proposed by Xiong et al in 2010 [19], adopts the hash Boolean matrix to polish up the calculation of intersection. The algorithm uses hash Boolean matrix to store the TID_set of itemsets. When calculating an intersection of two itemsets, it does not need to compare the TIDs of two itemsets one by one, but need to use bitwise 'and' operation on the two Boolean matrix. The technology of hash Boolean matrix can improve the efficiency only when the number of transactions of a database is not large. If the number of transactions is very large, hash Boolean matrix makes things worse.

Eclat-opt algorithm, proposed by Feng et al in 2013 [20], adopts technologies of double layer hash table, partition list of the set of itemset and TID lost threshold. These technologies can clip the candidate 3-itemset, reduce the search space as well as the time of generating candidate itemsets, and speed up the calculation of intersection. In general, Eclat-opt algorithm is much more effective than other Eclat based algorithms.

The existing improved Eclat algorithms can polish up the efficiency of original Eclat algorithm, but these algorithms still have some defects, such as large number of candidate itemset, inefficiency of itemsets connection and intersection.

3. Eclat_Growth Algorithm

According to the shortcomings of Eclat and existing improved algorithms, we proposed a new improved algorithm, called Eclat_growth, which is based on increased search strategy and vertical data format. In Eclat_growth, we established an increased two-dimensional pattern tree and the TID_sets of itemsets in the vertical data format table are added into the pattern tree row by row. New frequent itemsets are generated by combining the new added itemset with the existing frequent itemsets in the pattern tree. The process of combining new added itemset and existing frequent itemsets is based on the breadth-first search. Firstly, the candidate and frequent 2-itemsets are generated, then build up the candidate and frequent 3-itemsets, and so on, until all frequent itemsets in the pattern table are combined with the new added itemset. The candidate itemsets are composed directly by the frequent itemset and new added itemset, without the operation of itemsets connection. Due to the breadth-first search strategy, the prior knowledge can be used for clip the candidate itemsets completely. Thus, all redundant candidate itemsets will be clipped. Instead of intersecting the TID_sets of two itemsets, Eclat_growth adopts a new

method called BSRI (Boolean array setting and retrieval by indexes of transactions) to calculate the intersection and support degree of two itemsets. The technology of BSRI can obviously decrease the computational complexity.

3.1. The Main Process of Eclat_Growth Algorithm

The main process of Eclat_growth algorithm is shown in Table 3. First, it scans the database and stores it into a table using vertical data format. Second, it establishes a null increased two-dimensional pattern tree and adds the TID_sets of itemsets in the vertical data format table into the pattern tree row by row to generate new frequent itemsets. Finally, all frequent itemsets can be found by picking up all nodes in the pattern tree.

Table 3. The Main Process of Eclat_Growth

```
Function Eclat_growth-Algorithm (Database: D, Min_sup: MS)
// the basic function of Eclat_growth
Define: VM : Vertical Matrix, PT: Two-dimensional Pattern Tree,
     FIs: Frequent Itemsets
//Input: D, MS; Output: All FIs
L1. VM = CreateVMfromDatabase(Database: D)
L2. PT= CreateNullPatternTree
L3. for i = 1 to length(VM) do
L4.   if length(VM[i].TID_sets) >= MS then
L5.    AddItemsettoPatternTree(VM[i], PT, MS);
L6. end // end for i = 1 to length(VM)
L7. FIs=GetAllFrequentItemsetsfromPatternTree(Two-dimensional
          Pattern Tree: PT)
End
```

3.2. The Increased Two-Dimensional Pattern Tree

The increased two-dimensional pattern tree is composed of nodes and a layer pointer array. A node is defined as following: TreeNode={itemset, TID_set, PointerstoFathers, PointerstoChildren, HorizontalPointer, IsValid}. Every TreeNode includes the information of a frequent itemset, such as the itemset and the TID_set. Besides itemset and TID_set, a TreeNode also has other four elements, we give the illustration of the four elements as following: the element "PointerstoFathers" are pointers pointing to the nodes of its two fathers; the element "PointerstoChildren" are pointers pointing to the nodes of its children; the element "HorizontalPointer" are a transverse pointer, which is used for connecting the frequent itemsets with same length together; the element "IsValid" is a boolean value, which is used for indicating whether the node can be combined with another node. If an $(K+1)$-itemset A is joined by two k-itemset B and C, then B and C are the fathers of A, and A is the child of B and C. The pattern tree has multiple layers and the frequent itemsets with same length belong to the same layer. The layer pointer array is defined as following: LayerPointers: array of Pointer. The elements of layer pointer array are pointers pointing to the first node of all layers. So we can find all frequent *1*-itemsets based on the first element of the layer pointer array, and find all frequent *2*-itemsets based on the second element, and so on. All frequent itemsets can be found from the elements of the layer pointer array.

The first step of establishing the increased two-dimensional pattern tree is tree initialization: add a null node to tree as the father node of all frequent 1-itemsets, and set the length of LayerPointers as zero. After initialization, every TID_set of frequent 1-itemset in the vertical data format table are added into the pattern tree as the nodes of the first layer. Supposing the new added itemset as I_n, which contains the transactions of

TID_set (I_n), the process of adding a new itemset to pattern tree can be described as following.

Firstly, build a new node "Node(I_n)" for I_n. The elements of Node(I_n) are: Node(I_n).itemset=I_n, Node(I_n).TID_set=TID_set(I_n), Node(I_n).PointerstoFathers=nil, Node(I_n).PointerstoChildren=nil, and Node(I_n).IsValid=true. The HorizontalPointer of the last node in the first layer (frequent 1-itemset layer) is set to point to Node(I_n).

Combine Node(I_n) with every node (Node(I_{x-1})) in the first layer (frequent 1-itemset layer) to generate candidate 2-itemsets and calculate the support degree of the candidate 2-itemsets. If a candidate 2-itemset is frequent, a new node for the candidate 2-itemset will be built, the element "itemset" of new node will be set as the combination of I_{x-1} and I_n, and the element "TID_set" of new node will be set as the intersection of TID_set(I_{x-1}) and TID_set(I_n). Also, the element "PointerstoFathers" of new node will point to Node(I_{x-1}) and Node(I_n). Finally the HorizontalPointer of the last node in the second layer (frequent 2-itemset layer) is set to point to the new node. If a candidate 2-itemset is not frequent, it will need to be abandoned and the element "IsValid" of all children of Node(I_{x-1}) must be set as false.

Combine Node(I_n) with every node (Node(I_{x-2})) in the second layer (frequent 2-itemset layer) to generate candidate 3-itemsets. If the element "IsValid" of Node(I_{x-2}) is false, it cannot be combined with Node(I_n). Thus, we need to set the value of "IsValid" as true for further combination. If the element "IsValid" of Node(I_{x-2}) is true, it needs to be combined with Node(I_n) to generate a candidate 3-itemset, and to calculate the support degree of the candidate 3-itemset. If the candidate 3-itemset is frequent, a new node for the candidate 3-itemset will be built, the element "itemset" of new node will be set as the combination of I_{x-2} and I_n, the element "TID_set" of new node will be set as the intersection of TID_set(I_{x-2}) and TID_set(I_n). Also, the element "PointerstoFathers" of new node will point to Node(I_{x-2}) and Node(I_n), finally set the HorizontalPointer of the last node in the thrid layer (frequent 3-itemset layer) to point to the new node. If a candidate 3-itemset is not frequent, it needs to be abandoned and the element "IsValid" of all children of Node(I_{x-2}) must be set as false.

In the same way, Node(I_n) needs to be combined with all of the nodes in other layers to get all frequent itemsets.

The process of adding a TID_set of frequent 1-itemset in the vertical data format table to the pattern tree is shown in Table 4.

Table 4. The Process of Adding an Itemset Data to Pattern Tree

```
Function AddItemsettoPatternTree(Vertical Matrix: VM[i], Two-
                                 dimensional Pattern Tree: PT, Min_sup: MS)
// add a new node to Pattern Tree
Define: tmpNode, newNode, newCombineNode : TreeNode
//Input: VM, PT; Output: a new Pattern Tree
L1. for i = 1 to length(LayerPointers) do
L2.   if i =1 then
L3.     newNode = AddNewNodetoTree(VM[i])
L4.   tmpNode = LayerPointers[i]
L5.   while tmpNode ≠ null do
L6.     if tmpNode.isValid = true then
L7.       newCombineNode = CombineNewPattern(newNode,
                  tmpNode)
L8.       if Support(newCombineNode) >= MS then
L9.         AddNodetoTree(newCombineNode)
L10.      else
L11.        SetAllChildrenFalse(newCombineNode)
L12.      end //end if Support(newCombineNode) >= MS
```

L13. else
L14. tmpNode.isValid == true;
L15. end //end if tmpNode.isValid = true
L16. tmpNode = tmpNode.HorizontalPointer
L17. end //end while tmpNode ≠ null
L18. end //end for i = 1 to length(LayerPointers)
End

Taking the database shown in Table 2 as an example, by setting the minimum support degree of 3, the process of building the increased two-dimensional pattern is demonstrated in figures from Figure 3 to Figure 6. In the figures, the nodes in bold ellipses are frequent itemsets; the nodes in broken line ellipses are candidate non-frequent itemsets, which are not added into the pattern tree; the nodes in the ellipses with dark grey background are new added nodes in each step; the nodes in the ellipses with black background are unable to combine with the new added node in each step. The figures show that for the database in Table 2, there are 17 frequent itemsets are generated (eliminating the frequent 1-itemsets) and 8 frequent itemsets are generated. Comparing with Eclat (Figure 2), by adopting the technology of increased two-dimensional pattern tree, Eclat_growth can clip the candidate itemsets completely and reduce the number of candidate itemsets obviously under the prior knowledge.

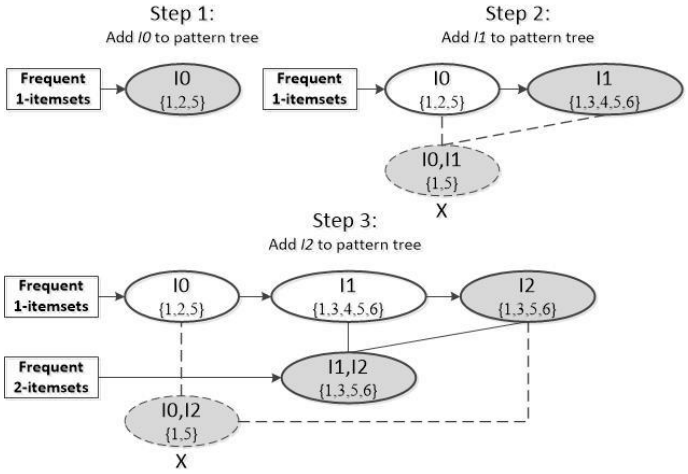


Figure 3. The Process of Building a Pattern Tree (Step1 to Step3)

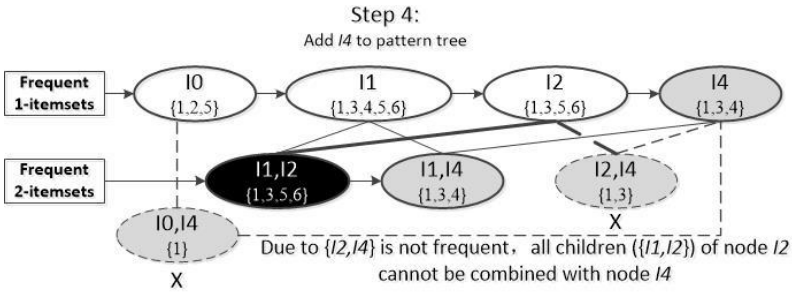


Figure 4. The Process of Building a Pattern Tree (Step4)

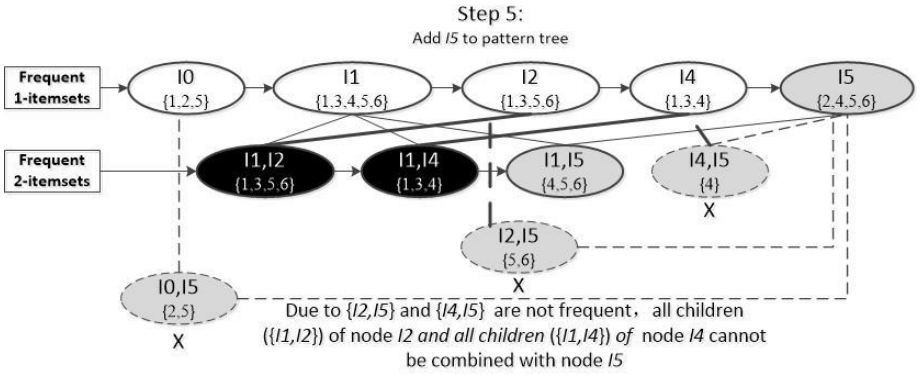


Figure 5. The Process of Building a Pattern Tree (Step5)

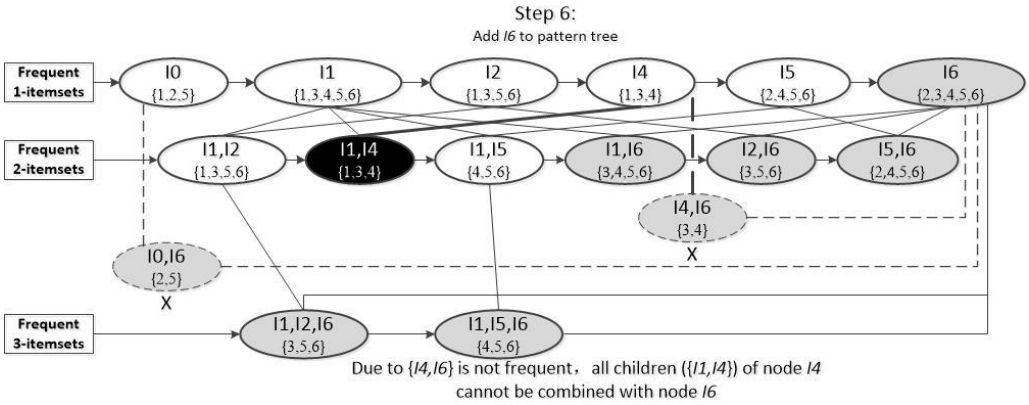


Figure 6. The Process of Building a Pattern Tree (Step6)

3.3. The Calculation of Intersection and Support Degree

The calculation of intersection is the most time-consuming step in Eclat. It needs to compare the TIDs of two itemsets one by one to evaluate which is ineffective when the number of TIDs is very large. Although making some improvements, the basic method for calculating intersection of the existing Eclat based algorithms is still the same with Eclat. In this paper, we proposed a high efficient method called BSRI (Boolean array setting and retrieval by indexes of transactions) to calculate the intersection and support degree of two itemsets. In the follows, we introduce the method of BSRI sophisticatedly.

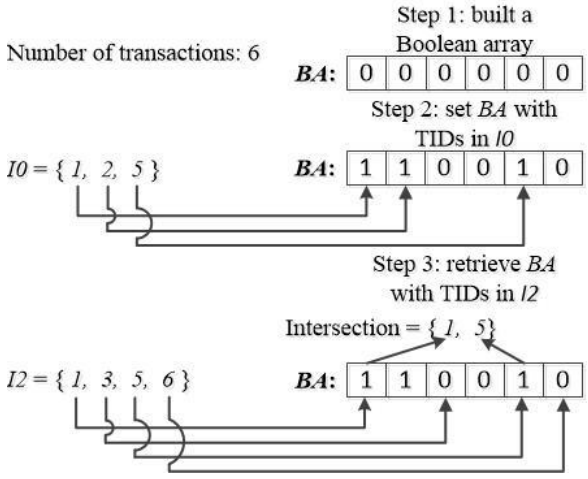


Figure 7. The Process of BSRI

We establish a Boolean array, set the length of array as the number of transactions, and set all elements of the array as 0. Taking the TIDs in an itemset as index, the corresponding elements of Boolean array are set as 1. Then we retrieve the elements of Boolean array according to the TIDs in another itemset. If an element is 1, the according TID into intersection will be add and the support degree will be increased. For example, we need to calculate the intersection and support degree of *I0* and *I2* in the database shown in Table 2, *I0*={*1,2,5*}, *I2*={*1,3,5,6*}. Since the number of transactions of the database is 6, the Boolean array is built as *BA*=(0,0,0,0,0,0). Based on the TIDs in *I0*, we set the indexed elements of *BA* as 1, *BA*[1]=1, *BA*[2]=1, *BA*[5]=1, now *BA*=(1,1,0,0,1,0). Based on the TIDs in *I2*, we retrieve the corresponding elements of *BA*, we get *BA*[1]=1, *BA*[3]=0, *BA*[5]=1, *BA*[6]=0, so the intersection if *I0* and *I2* is {*1,5*} and the support degree is 2. The above process is shown in Figure 7. The above introduction shows that by using the technology of BSRI, we only need to set and search the elements of a Boolean array to calculate the intersection and support degree.

In the process of building an increased two-dimensional pattern tree, we use the TIDs of the new added itemset for setting the Boolean array and use the TIDs of the frequent itemset in pattern tree for retrieving. Thus the efficient can be enhanced further.

4. Experimental Studies

In order to present the efficiency of the Eclat_growth algorithm, we compare it with the original Eclat algorithm [16], the Eclat_Diffsets algorithm [18], the hEclat algorithm [19] and the Eclat-opt algorithm [20]. The comparisons of Eclat with other algorithms are detailedly presented in [16] and [17]. In this paper, we only compare Eclat_growth with the Eclat based algorithms. All experiments were performed on a PC with Inter Core 1.8G and 4G main memory, running on Microsoft Windows 8 64-bit and all the programs are coded in Pascal. We apply the algorithms on six synthetic databases which have been commonly used for many association rules algorithms. Among the databases, two of them are sparse ones, and three are dense ones, the numbers of transactions are from 3000 to 990002, the numbers of items are from 72 to 41271. All databases can be found online [21]. Table 5 shows the information of the databases.

Table 5. The Information of Test Databases

Database	Number of transactions	Number of items	Average length	type
T10I4D100K	100000	1000	10	Sparse
kosarak	990002	41271	7	Sparse
T40I10D100K	100000	1000	40	Mid-dense
mushroom	8124	120	23	Dense
chess	3196	76	37	Dense
accidents	340183	469	34	Dense

4.1. Experimental Results

In the first experiment, we investigate the run time of the Eclat_growth algorithm comparing with Eclat, hEclat and Eclat-opt based on two sparse database: T10I4D100K and kosarak. The test results are shown in Figure 8 and Figure 9. Since Eclat_Diffsets is suitable for dense database, we do not contain Eclat_Diffsets in this experiment.

In the second experiment, we investigate the run time of the Eclat_growth algorithm comparing with Eclat, Eclat_Diffsets, hEclat and Eclat-opt based on three dense database: T40I10D100K, mushroom and chess. The test results are shown in Figure 10, Figure 11 and Figure 12.

In the third experiment, we investigate the run time of the Eclat_growth algorithm comparing with Eclat, Eclat_Diffsets, hEclat and Eclat-opt by varying the number of transactions on the basis of databases of kosarak and accidents. We mine the frequent itemset from kosarak by taking the number of transactions from 200000 to 990000 and the minimum support is set as 0.3%. For accidents, we set the minimum support as 50%, and take the number of transactions from 100000 to 340000. The test results are shown in Figure 13 and Figure 14.

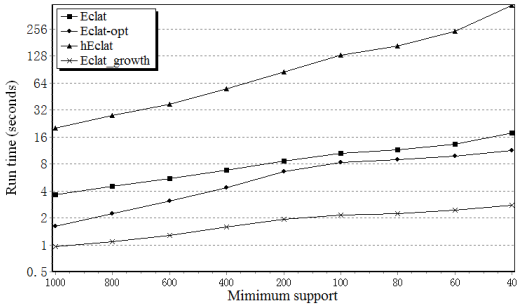


Figure 8. Run Time on T10I4D100K

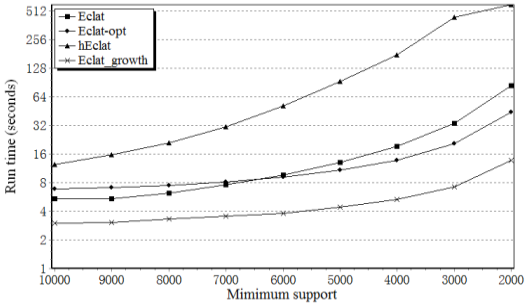


Figure 9. Run Time on Kosarak

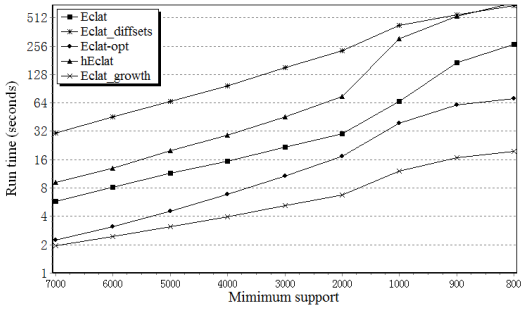


Figure 10. Run Time on T40I10D100K

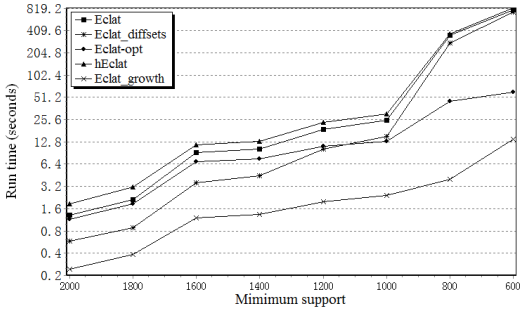


Figure 11. Run Time on Mushroom

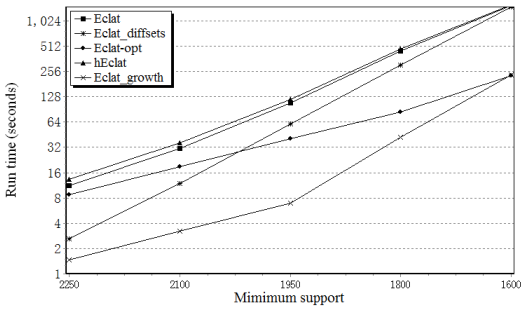


Figure 12. Run Time on Chess

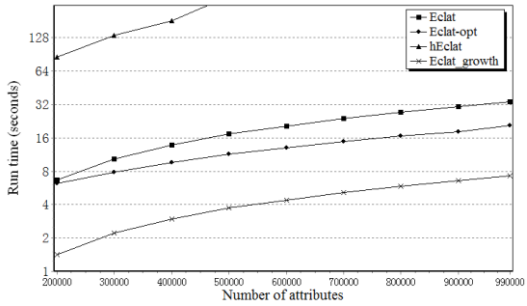


Figure 13. Run Time on Kosarak with Variant Number of Transactions

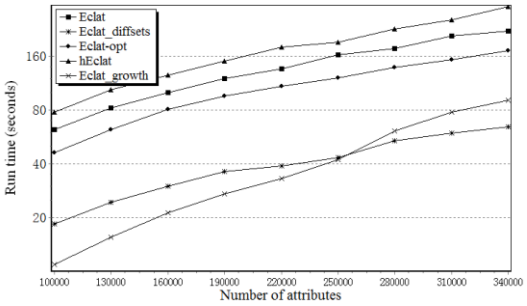


Figure 14. Run Time on Accidents with Variant Number of Transactions

4.2. Performance Analysis

According to the results in the first experiment, for mining associate rules on sparse database with large transactions (100000 and 990002) and large items (1000 and 41271), Eclat_growth is more effective than Eclat-opt and Eclat, however, hEclat is the most time-consuming. Taking T10I4D100K as example (Figure 8), when the minimum support varies from high support (1000， 1%) to low support (40， 0.04%), the run time of Eclat_growth is 58% versus 24% of that of Eclat-opt, and is 26% versus 15% of that of Eclat. In the second experiment, for the mid-dense database of T40I10D100K (Figure 10), Eclat_growth also has the highest efficiency. For the dense database of mushroom (Figure 11), when the minimum support varies from high support (2000， 25%) to low support (600， 7.5%), the run time of Eclat_growth is 13% versus 23% of that of Eclat-opt, and is 18% versus 1.8% of that of Eclat.

The performances of Eclat based algorithms mainly depend on the number of candidate itemsets and the efficiency of calculating intersection. The methods of generating candidate itemsets in Eclat, Eclat_Diffsets and hEclat are all the same. In the generation of candidate itemsets, there are no clip strategies under the prior knowledge, therefore, many redundant candidate itemsets will be built and calculated. Double layer hash table is

used in Eclat-opt to clip the candidate itemsets. Because the Double layer hash table can only work in generating candidate 3-itemsets from frequent 2-itemsets, the performance is very limited. Due to the increased two-dimensional pattern tree, the prior knowledge can be used in Eclat_growth for clip the candidate itemsets completely and all redundant candidate itemsets will be clipped. Taking the database of T10I4D100K and mushroom for example, we compare the numbers of candidate itemsets of Eclat and Eclat_growth when mining association rules with different minimum supports and the results are shown in Table 6 and Table 7 which demonstrate that the numbers of candidate itemsets of Eclat_growth are obviously less than that of Eclat, especially when the database is sparse. Less candidate itemsets is an important factor for high performance of Eclat_growth.

Table 6. The Numbers of Candidate Itemsets for Mining T10I4D100K

Support	400	200	100	80	60	40
the numbers of candidate itemsets for Eclat	398998	585042	806403	950397	1258847	2168909
the numbers of candidate itemsets for Eclat_growth	199562	286621	372228	432736	581053	1091003
the numbers of frequent itemsets	2001	13255	27532	35019	44583	62864

Table 7. The Numbers of Candidate Itemsets for Mining Mushroom

Support	1600	1400	1200	1000	800	600
the numbers of candidate itemsets for Eclat	59528	68388	118273	155182	724498	1186352
the numbers of candidate itemsets for Eclat_growth	55537	62357	105383	131500	598489	1016465
the numbers of frequent itemsets	53583	58773	98575	118455	574431	932181

Besides the number of candidate itemsets, the efficiency of calculating intersection is another important factor for the performance of Eclat based algorithms. Eclat and Eclat_Diffsets need to compare the TIDs of two itemsets one by one to calculate the intersection. Eclat-opt adopts the same method as Eclat, but it uses an optimized strategy called TID lost threshold to improve the performance of calculating intersection to a certain degree. hEclat adopts hash Boolean matrix to polish up the calculation of intersection, which can improve the efficiency only when the number of transactions of a database is not large. If the number of transactions is very large, hash Boolean matrix makes things worse. Eclat_growth adopts BSRI to calculate the intersection and support degree of two itemsets, it can effectually improve the performance by less operations than that of other Eclat based algorithms. We test the performance of BSRI by using BSRI in Eclat algorithm. Table 8 shows the run time of the original Eclat algorithm and the Eclat with BSRI for mining association rules on T40I10D100K, the minimum support is set from 7000(7%) to 900(0.9%). The test result shows that BSRI can obviously improve the performance. For the database of T40I10D100K, 34% performance is enhanced on average.

Table 8. The Run Time of Eclat and Eclat with BSRI for Mining T40I10D100K

Support	7000	6000	5000	4000	3000	2000	1000	900
Eclat	5.82	8.20	11.49	15.45	21.62	29.87	66.73	169.64
Eclat with BSRI	3.94	5.25	7.09	9.25	12.63	17.59	45.87	147.53

In the third experiment (Figure 13 and Figure 14), we investigate the effect of transactions number on the algorithms' performances by mining association rules from kosarak and accidents with different transactions numbers. According to the experimental results, the transactions number has a serious influence on the efficiency of the five algorithms. As the number of transactions becoming larger and larger, the run time of five algorithms will increase multiply. Comparing with other algorithms, the efficiency of Eclat_growth is most obviously cut down. Due to the increased search strategy, the whole pattern tree needs to be stored in memory. When the number of transactions is very large, the memory usage is huge, which cuts down the performance of Eclat_growth obviously. For other algorithms, the memory usages are evidently less than Eclat_growth, because they only need to store two different length frequent itemsets in memory. In Figure 14, when the number of transactions is larger than 250000, Eclat_growth is less efficient than Eclat_Diffsets. However, in other conditions, Eclat_growth is still faster than other algorithms.

5. Conclusions

Mining association rules is one of the most important problems in data mining. Therefore, many algorithms were proposed to facilitate mining association rules. Although Eclat algorithm has the high performance, it still has some defects, such as large number of candidate itemset, inefficiency of itemsets connection and intersection. According to the shortcomings of Eclat, we proposed an improved Eclat algorithm called Eclat_growth. Eclat_growth algorithm, on the basis of increased search strategy, adopts two new technologies, such as increased two-dimensional pattern tree and BSRI. In the process of building an increased two-dimensional pattern tree, the prior knowledge can be used for clipping the candidate itemsets completely and all redundant candidate itemsets will be clipped. BSRI is used for calculating the intersection and support degree, it can obviously decrease the computational complexity. The theoretical analysis and experimental studies all indicate that Eclat_growth has the highest performance in mining associating rules from various databases than Eclat, Eclat-diffsets, Eclat-opt and hEclat.

Acknowledgements

This work was supported by the National Science Foundation of China (No. 51305213), the Scientific Research Fund of Zhejiang Provincial Education Department (Y201224583) and the K. C. Wong Magna Fund in Ningbo University.

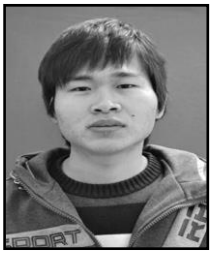
References

[1]. R. Agrawal, T. Imilienski and A. Swami, "Mining association rules between sets of items in large databases," Proceeding of the ACM SIGMOD Int'l Conference on Management of Data, Washington DC, **(1993)**, pp. 207-216.
[2]. J. Han, J. Pei and Y. Yin, "Mining frequent patterns without candidate generation", Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, **(2000)**, pp. 1-12.
[3]. J. Dong and M. Han, "BitTableFI: An efficient mining frequent itemsets algorithm", Knowledge-Based Systems, vol. 20, no. 4, **(2007)**, pp. 329-335.
[4]. C. Aflori and M. Craus, "Grid implementation of the Apriori algorithm", Advances in Engineering

Software, vol. 38, no. 5, **(2007)**, pp. 295-300.
[5]. F. Berzal, J. Cubero and N. Marin, "TBAR: An efficient method for association rule mining in relational databases", Data & Knowledge Engineering, vol. 37, no. 1, **(2001)**, pp. 47-64.
[6]. D. C. Pi, X. L. Qin and N. S. Wang, "Mining Association Rule Based on Dynamic Pruning", Mini-Micro Systems, vol. 10, **(2004)**, pp. 1850-1852.
[7]. J. Pork, M. Chen and P. Yu, "An effective hash based algorithm for mining association rules", ACM SIGMOD, vol. 24, no. 2, **(1995)**, pp. 175-186.
[8]. S. Brin, R. Motwani and C. Silverstein, "Beyond market baskets: generalizing association rules to correlations," ACM SIGMOD Conference on Management of Data, Tuscon, AZ, (**1997**), pp. 265–276.
[9]. H. F. Li and S. Y. Lee, "Mining frequent itemsets over data streams using efficient window sliding techniques", Expert Systems with Applications, vol. 36, no. 2, **(2009)**, pp. 1466-1477.
[10]. Y. F. Zhong and H. B. Lv, "An Incremental Updating Algorithm to Mine Association Rules Based on Frequent Pattern Growth", Computer engineering and Application, vol. 26, **(2004)**, pp. 174-175.
[11]. R. Balazs, "Nonordfp: An FP-Growth Variation without Rebuilding the FP-Tree", Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementa-tions, **(2004)**.
[12]. G. Gosta and J. F. Zhu, "Fast Algorithms for Frequent Itemset Mining Using FP-Trees", IEEE Transactions on Knowledge and Data Engineering, vol. 17, no. 10, **(2005)**, pp. 1347-1362.
[13]. W. Ke, T. Liu, H. J. Wei and L. J. Qiang, "Top down fp-growth for association rule mining", The 6th Pacific-Asia Conference, PAKDD 2002, Taipei, Taiwan, **(2002)**, pp. 334-340.
[14]. J. Pei, J. W. Han and H. J. Lu, "H-Mine: Fast and space-preserving frequent pattern mining in large databases", IIE Transactions, vol. 39, no. 6, **(2007)**, pp. 593-605.
[15]. K. C. Lin, I. Liao and Z. S. Chen, "An improved frequent pattern growth method for mining association rules", Expert Systems with Applications, vol. 38, no. 5, **(2011)**, pp. 5154-5161.
[16]. M. Zaki, "Scalable algorithms for association mining", IEEE Transactions on Knowledge and Data Engineering, vol. 12, no. 3, **(2000)**, pp. 372-390.
[17]. L. S. Thieme, "Algorithmic Features of Eclat", Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, Brighton, UK, November, **(2004)**.
[18]. M. J. Zaki and K. Gouda, "Fast vertical mining using diffsets", Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, New York, USA, **(2003)**, pp. 326-335.
[19]. X. Z. Yang, C. P. En and Z. Y. Fang, "Improvement of Eclat algorithm for association rules based on hash Boolean matrix", Application Research of Computers, vol. 27, no. 4, **(2010)**, pp. 1323-1325.
[20]. F. P. En, L. Yu, Q. Q. Ying and L. L. Xing, "Strategies of efficiency improvement for Eclat algorithm", Journal of Zhejiang University (Engineering Science), vol. 47, no. 2, **(2013)**, pp. 223-230.
[21]. Available at: http://fimi.ua.ac.be/data.

Authors

Zhiyong Ma, received the Ph.D. degree in Institute of Mechanical Design, Zhejiang University in 2010. He is working in the Faculty of Mechanical Engineering and Mechanics, Zhejiang Provincial Key Lab of Part Rolling Technology, Ningbo University, Ningbo, China. His current research interests include mechanical symmetry theory and knowledge discovery in mechanical design.



Juncheng Yang, is now pursuing a master degree in the Faculty of Mechanical Engineering and Mechanics, Zhejiang Provincial Key Lab of Part Rolling Technology, Ningbo University, Ningbo, China. His current research interest is knowledge discovery in mechanical design.

Taixia Zhang, is now pursuing a master degree in the Faculty of Mechanical Engineering and Mechanics, Zhejiang Provincial Key Lab of Part Rolling Technology, Ningbo University, Ningbo, China. Her current research interests include mechanical symmetry theory and knowledge discovery in mechanical design.

Fan Liu, is now pursuing a master degree in the Faculty of Mechanical Engineering and Mechanics, Zhejiang Provincial Key Lab of Part Rolling Technology, Ningbo University, Ningbo, China. Her current research interests include mechanical symmetry theory and knowledge discovery in mechanical design.