ATLANTIS PRESS

Advances in Computer Science Research, volume 58

Modeling, Simulation and Optimization Technologies and Applications (MSOTA 2016)

The Research and Improvement Based on FP-Growth Data Mining Algorithm

Quanzhu Yao, Xingxing Gao*, Xueli Lei and Tong Zhang

666 mailbox , School of Computing , Xi 'an University of Technology , No.5.Jinhua South Road , Beilin District , Xi 'an City, Shaanxi Prov,China

*Corresponding author

***Abstract*—In order to improve the availability in space domain and raise time efficiency of the algorithm, increasing more FP-array to decrease the traversing of FP-tree in this paper. It has put forward to applying the node switching strategy to generate more packed FP-tree. It has come up with increasing more FP-array to decrease the traversing of FP-tree. The space of storing FP-tree would be reduced, it could avoid allocating more other space and improve the utilization factor of the space. The solution indicates that the efficiency of time and space of improved FP-Growth algorithm is higher than that of classical FP-Growth algorithm and TFP-growth algorithm. What is new and original in this paper is it applies node switching strategy to generate more packed FP-tree and increasing more FP-array to decrease the traversing of FP-tree.**

Keywords-FP-Growth algorithm; data mining, frequent itemsets; FP-tree, ENFP-growth algorithm

I. INTRODUCTION

Data Mining is also known as data discovery technical, which is covered broadly. It covers clustering, classification and association rules. It is applied to seek significant information which is hid in the large-scale data set. Since the application value of meaningful data in the modern business is increasingly important, data mining is studying widely and applying to areas such as science discovery, marketing, fraud detection, supervising and so on. Association rules mining is used to association analysis of data mining. It is an efficient and popular method in extracting meaningful data from big data set recently. It could find efficient association relation among items in big data set. Frequent itemsets mining is the first and central step in the process of association rules mining. The classical association rules algorithm includes apriori algorithm, FP-Growth algorithm and other mining algorithms which is based on frequent itemsets of matrix.

II. RELATED WORK

FP-Growth algorithm is an improved algorithm proposed by J.Han and so on. They aim at the defects such as Apriori which needs to traverse database frequently, generates a large number of candidate itemsets. FP-Growth algorithm has common adaptation to the rules of different lengths. It could compress the information of database to frequent tree. And it has significant enhancement in efficiency compared with previous Apriori algorithm[1].However,FP-Growth algorithm also has defects of using a large amount of data to build huge amount of FP-tree. It cost a mass of space and time, and the recursion algorithm has low efficiency, so the mining efficiency is low. Many researchers put forward a lot of improved method based on the FP-Growth algorithm. H-mine algorithm stores items by applying arrays nor tree. It has good efficiency under the conditions of sparse database. Christian Borgelt[2] has come up with Relim[3] algorithm which is based on FP-growth theory. This algorithm does not need to generate candidate itemsets, it has simple structures, high space utilization and is easily realized. But it has low advantage on the handling amount of transaction which is based on the pointer. Many scholars put forward the view of the defects of the parallel algorithm for mining. It has put forward ENFP-growth algorithm to develop more efficient FP-growth improvement algorithm.

A. Background

1) FP-growth algorithm

FP-growth algorithm could be divided into two stages: building FP-tree and mining frequent itemsets[4] from FP-tree. It needs to scan database twice to build FP-tree. At first, it scans database at a time to generate frequent 1-itemsets and to order them according to the descending order, then puts them to L-link. It scans database again, one tuple of database corresponding to the relation and items frequency, then it put these information into FP-tree. FP-growth algorithm could use the conditions of model base to build a little FP-tree which is called condition FP-tree[5][6]. This is an iterative process until there is no conditions of model base could be generated.

Since FP-growth algorithm does not need to scan database many times and does not generate the candidate itemsets. It becomes the broadest association rules algorithm at present. But there also exists some defects, mainly displays in:

(1) It is not enough to the compression of the transaction. It analyzes the process of FP-tree structure, and compares the transaction and the node which is the existing branch path when build FP-tree. If the transaction and the FP-tree share the prefix, then plus 1 to the count of node of prefix of FP-tree, otherwise build a new node. So when it ignores the comparison of the suffix node, it could not compress the suffix node. Then it could generate too many branches, waste memory resources, prolong the time of recursive search and reduce the efficiency of FP-tree mining.

(2)To scan FP-tree many times will cost time. FP-growth algorithm has two processes to waste time: First, building FP-tree; second, traversing FP-tree to mine frequent itemsets. And the second wasting time part could be divided as two

parts: First, traversing FP-tree to generate frequent 1-itemsets and counting the supporting number of frequent 1-itemsets. Second, traversing FP-tree to mine frequent itemsets[7].

(3)To store condition FP-tree needs allocating other space. It finds that there are two statistics by elaborate analysis of primary algorithm: 1.the process of item is to proceed the item of header table from the backward. That is to say, it could generate the condition FP-tree of the item from the backward. 2. The item is forward mapping, so the process and the mapping of item of the primate algorithm are from the backward. If the information of new generated condition FP-tree is saved in the primate FP-tree, and new condition FP-tree does not save the information, there would be conflict. Since: the primate algorithm proceeds the items from backward, when an item is proceeding, the item before it has not been proceeded; and the mapping to item of the primate algorithm is from backward, when it proceeds some items, it needs to get the information of the item before itself in the condition of the item exists to make up with the relevant item. But they have not been proceeded, there would not update the primate information, otherwise when it proceeds the information of the item primate tree, there would exist mistake.

B. The Improvement of FP-growth Algorithm

For the defects of FP-growth algorithm analyzed above, the solution would be put forward in this paper. As the defect (1) puts forward to node switching method to compress and store FP-tree. As the defect (2) puts forward to reduce traversing FP-tree by Auxiliary matrix[8]. As the defect (3) puts forward to proceeding the items in the header table from the bottom up and proceeding the items once upon a time back. In the following sections it would detail each improvement methods respectively.

1) Improved FP-tree structure

FP - growth algorithm can be divided into two steps, the first step is to construct of FP frequent pattern tree FP- tree. The second step is to mine frequent itemsets from the frequent pattern tree. When it analyzes FP - tree construction process and contrasts of transaction and FP - tree node of existing branch path, if the transaction item and FP – tree share prefix, then the count will plus 1 in the prefix.

That is only under the condition of the same prefix can be compressed. If the prefix is not same, even nodes have the same suffix also need to establish a branch, cannot be compressed. If the transaction has many of the same node prefix, so build a compact FP – tree, the best situation is to compress into a branch, this could save memory greatly. But if the current nodes is not same and has the same suffix transaction, if only consider the current node are the same, it must set up branches, also. Then it constructs sparse FP - tree in memory. So it ignores the contrast of suffix node and cannot compress the suffix nodes. Then there could be too many branches, it not only wastes of memory resources, also lengthens a recursive search time lengthen, results in lower efficiency of FP - tree mining.

According to this, it could build the FP-tree method to construct compact FP–tree from improve FP-tree. The classical method of building FP-tree way is as same as the way of tree in nature. The higher the frequent degree is, the closer the roots of nodes is. The lower the frequent degree is, the further the roots of nodes is. So the frequent counting from the roots to the leaves is from high to low. By analyzing the structure and structural features of FP-tree in this paper, in order to construct more compact FP-tree, it puts forward the ENFP-tree (Exchange Node FP-tree) to improve the utilization rate of memory and reduce the searching time. When the transaction is not at the same as the FP - tree current nodes, then continues to compare with the subsequent nodes. If it is as same as the subsequent node and meets the exchange, then exchange of the FP - tree current nodes with the followed one. Here the exchange nodes are for the same tree. The purpose is to construct the trunk, and more compact FP – tree as possible. It could achieve the objective of the whole optimization by the local optimization strategy to improve the utilization rate of memory.

a) The improved method. In the process of the exchange node constructing the FP – tree, the following questions need to be considered: When it could exchange nodes; What conditions are to exchange nodes; after node exchange if it could guarantee the original database without loss or bringing in a new item, that is to say if it could guarantee the integrity of database.

Theory 1: If the current node has no fewer than two children nodes, the node could not exchange with its son node. Prove: to the transaction $T1=\{e_1,e_2,\cdots e_i,\cdots,e_j,e_n\}$,transaction $T2=\{h_1,h_2,\cdots h_i,\cdots,h_j,\cdots h_n\}$ and $T3=\{g_1,g_2,\cdots g_i,\cdots,g_k\cdots g_n\}$. If the former i items of the transaction T1 is as same as the transaction T2’ s, but the i + 1 item are different. That is, $e_p=h_p$ (p (1…i)),$e_{p+1}\neq h_{p+1}$. Then there will be two branches in the process of structure tree $\{e_i, e_{i+1}\}$, $\{e_i, h_{i+1}\}$. As for T3 , if the former i-1 items of T_2’ s and $g_i = h_{i+1}$, if exchange h_i，h_{i+1}, then it will generate situations as following:

①e_{i+1} points to exchanged h_{i+1}, this will introduce items h_{i+1} into T_1 and lose items e_i , that is it will not be able to maintain the integrity of the original T_1 things after nodes exchanged.

②If it points to exchanged e_i, there will also be the T_1 in the introduction a more h_{i+1} and generate dirty data.

Above all, e_{i+1} points to other nodes could cause changes of the transaction of the primate database. Therefore, if the current node has no less than two child nodes, it is not feasible to exchange node.

Prove: to the transaction $T_1=\{e_1,e_2,\cdots e_i,\cdots,e_j,\cdots e_n\}$,the transaction$T_2=\{h_1,h_2,\cdots h_i,\cdots,h_j,\cdots h_n\}$ and $T_3=\{g_1,g_2,\cdots g_i,\cdots,g_k,\cdots g_n\}$. If the former i items of the transaction T1 are as same as the transaction T2’s, but i + 1 items are different. That is, $e_p=h_p$（p (1…i)）,$e_{p+1}\neq h_{p+1}$. As for T_3 , if the former i-2 items of T_2’ s and $g_{i-1}\neq h_{i-1}$, $g_{i-1}=h_i=e_i$, there will be two branches in the process of constructing$\{e_i,e_{i+1}\}$,$\{e_i,h_{i+1}\}$. It could exchange e_i and e_{i-1}, there will be three branches $\{e_{i-2},e_i,e_{i-1},e_{i+1}\}$,$\{e_{i-2},e_i,e_{i-1},h_{i+1}\}$,$\{e_{i-2},e_i,g_i,g_{i+1}\}$. To the transaction T_1, T_2 just changes the order of the items in the transaction, and introduce no other information. When it

generates tree according to the condition model base, it checks the statistics of all nodes on the conditions of model base. It will not neglect information and its meaning remain.

Theory 3: On the whole, the closer to the root node, the higher support degree is in the tree.

Prove: To X={e_1,e_2,…e_i,…,e_j,…,e_n}, it is a branch of a tree, to e_i, e_j, e_j is closer to the root node than e_i, m is the support number of ej and n is the support number of e_i, then n+α (the threshold value)>=m. That is in the sequence, the support degree of the item which is close to the root node is larger.

In the proof process of theorem 2.3 with threshold value to control the switching frequency of the tree, the node could be exchanged only within the scope of the threshold value. If it surpasses the scope of the threshold value, even the same nodes will not continue to exchanging. On one hand this measures could limit comparison frequency, on the other hand, it also makes the support degree lager when the tree is closer to the root by threshold in the macro.

2) FP-array's raising

a) The analysis of FP-growth algorithm. (1)The core of FP-growth algorithm: scan the original database and construct FP-tree, then traverse the FP-tree and construct new conditions of FP-tree mining frequent itemsets. To scan FP-tree many times will spend time. FP-growth algorithm has two processes to waste time: First, building FP-tree; second, traversing FP-tree to mine frequent itemsets. And the second wasting time part could be divided as two parts: First, traversing FP-tree to generate frequent 1-itemsets and counting the supporting number of frequent 1-itemsets. Second, traversing FP-tree to mine frequent itemsets. Traversing the FP-tree is time consuming, so if the traversal times of FP-tree can be reduced, it will reduce the time consumption of the algorithm.

b) The improvement ideas of the algorithm. After the enough analysis of the FP-growth algorithm, in order to reduce the traversal times of FP-tree, it puts forward an auxiliary matrix to reduce the traversal times of FP-tree. To each item of the item header table T1 of condition FP-tree, in order to construct new condition FP-tree T1, it needs traverse FP-tree T1 twice. It could find all the frequent items in the conditions of model base of the item in the first traversing and construct the item header table of conditions FP-tree. If it constructs the condition FP-tree T1 and constructs a matrix A1 corresponding to a frequent item at the same time. As an attribute of the T1, it could find all frequent items directly by A1 and omit the first traversing of tree T1. Due to omitting the tree traversal for the first time in the running process of iteration algorithm, it enhances the time efficiency of the algorithm.

c) Construct FP-array. Theory 2.1: Suppose T used to represent the condition FP – tree, I = {i_1, i_2, …, i_n} is the item of item header table. The relationship among the frequent items could be showed with a (n-1)*(n-1) matrix. The count of support degree of frequent item pair is the element of the matrix.

Because of the matric is symmetrical, so the frequent item (i_j,i_k)and (i_k,i_j) denote a frequent item pair. It allows to store only one pair, and could store all the information with an inverted triangle matrix.

C. Change the Direction of the Item of the Map

1) The process characteristics of the FP-tree

To analysis carefully of the primitive algorithm, it finds that there are two characteristics of the primitive algorithm: 1.the process of item is to proceed the item of header table from the bottom up, and generates each of the sub-problem of the item in turn 2.The item is forward mapping, so the process and the mapping of item of the primate algorithm are from backward. So, it will be conflict if the information of new condition of FP-tree is stored in the original FP-tree, and is not saved in the new condition of FP-tree.

If the information of new condition of FP - tree is stored in the original FP-tree, it could cover the original information of items which has not been handled in a FP-tree. But these items are not processed at this point and are not be allowed to be covered. So the FP-growth algorithm could separate the space to store the new condition FP-tree. Because FP- growth algorithm is a recursive implementation, the proceeding of table is from down to up, the process of item is backward. Such a series of condition FP-tree are unavoidable.

2) Improved treatment of FP-tree

Through the analysis of the causes of above, it could get improvements from algorithm: Make the original algorithm to table processing order and to item opposite. It means to adjust the direction of the original algorithm of mapping, and into "for items from backward and mapping forward" (Approach is showed in figure3). With this scheme, when an item is processed, the following items have been processed, the item in front has not yet been processed, and because the item generates conditions FP-tree backward, the new conditions of FP-tree information could directly cover the behind item information. Because the SE items have been processed, and these information is no longer needed. It will not divide new space to store new condition FP-tree. And according to the plan to generate frequent itemsets as long as always. All of the behind information is stored in the original tree.

D. FP-growth Algorithm Improvement

1) Algorithm thought and pseudo code

2.1, 2.2, 2.3 at the fore has improved FP - growth algorithm from different angles. In this section, the previous three next improvement methods could be fusion to form a new improved algorithm ENFP FP-growth, improvement algorithm (Exchange Node FP-growth).

The pseudo code of ENFP-growth algorithm is as follows:

Input: The transaction database D, Minimum Support min_sup
Output: Frequent itemsets
call algorithm 1 to construct corresponding ENFP-tree,FP-array; Ahead-mining(ENFP-tree)to transaction database D. for process the item I_i（i=n~1）of header table L of ENFP-tree from the back forward
{
(3)Generate frequent 1–itemsets: M= {I_i|M.support=I_i.support}
(4) If(there are items behind I_i){
(5) If all the items behind I_i are proceeded {

(6)Map_back(ENFP-tree,Ii);
(7) Increase_back(ENFP-tree,M,i);
(8) }
(9) Else
(10) Call algorithm 3
(11) }
(12) }
Increase_back（ENFP-tree,M,i）
(13) for check the item Ij(j=n~i+1)of header table L of ENFP-tree from the back forward
(14) if (Ij.support>=minsup){
(15)generate frequent itemsets:M=(M ∪ {Ij}|M.support=Ij.support)
(16) if (there are frequent items behind Ij){
(17) Map_back (ENFP-tree,Ii);
(18) Increase_back (ENFP-tree,M,i);
(19) }
(20) }
Map_back (ENFP-tree,Ii)
(21) For the item Ij (j=i+1~n) behind Ii in header table L
(22) According to the corresponding support number of item in FP-array update support number of item of the header table item. /* If there is no support number of item which is behind Ii in FP-array, it indicates the item is no-frequent item and could be neglect.

Algorithm 1

ENPFP-tree, FP-array construct the specific process of the algorithm is as follows:

Input: The transaction database D, Minimum support minsup, Threshold α.

Output: ENFP-tree TFP-array

Step 1: Scan the transaction database D for the first time, calculating the support of each item count, remove the support number of item less than the minimum support, generate frequent 1-itemsets, and construct header table L With frequent 1-itemsets, and order the support degree in descending order.

Step 2: IF for the value of location I of the item Ii in the header table L ((len+10)/10<i<n), len is the whole length of the item header table.

Step 3: construct the root node of the ENFP-tree, and tag with "null". Each of the transaction in D scans the frequent itemsets of the transaction, then order the frequent itemsets in decreasing order in L. Tag the ordered frequent itemsets as [p|P]. And p is the first element, P is the table of left elements. Then call insert_tree([p|P],T),the proceeding is as following :

If T has child h, make h.item_name = p.item_name, then the count of h plus one;

Otherwise, select node e behind p, and node f behind h, judge h.item_name = e.item_name, f.item_name = p.item_name,or f.item_name = e.item_name three situations.

If there is one equaling in (2) and satisfies |e.count – p.count| <α (Introducing the threshold value of α representative to support degree of difference between the replacement items), and h only has one child node. Then the corresponding value of count pluses 1.

If it does not satisfy, it needs to construct a new node q, and pluses one to count, points the new node to its parent node. When P is not null, call the process of insert_tree ([p|P]) recursively.

Else call insert_tree2 ([p|P])

Step 3: call step 2 to construct FP-array

Insert_tree2 ([p|P], T)

(1) IF (T has child which makes h .item name=p. item name) THEN h plus one;
(2) ELSE construct a new node h, and set the count to be 1;
(3) If P is not null, call Insert_tree2 ([p|P], h) recursively.

Algorithm 2:

FP-array implement algorithm

(1)Construct an n*n triangular matrix with frequent 1-itemsets.
(2) When insert a transaction, if there exists two combination of frequent items, the value of count of combination of frequent items pluses one.

Algorithm 3:

Based on ENFP-tree,FP-array,backward mapping divide new space to store FP-growth[9] improved algorithm of condition FP-tree.

It calls LENFP-growth algorithm which is based on ENFP-tree, FP-array, backward mapping dividing new space to store FP-growth improved algorithm of condition FP-tree. LENFP-growth algorithm constructs tree with ENFP-tree strategy in the phrase of constructing tree. And use FP-array and backward mapping in the phrase of mining tree, but it needs to divide other space to store condition FP-tree strategy.

The idea of LENFP-growth algorithm is as following:

Mine frequent itemsets with the order of support degree increasing according to L. To each frequent item, find the item and condition mode based item behind it according to FP-array , then construct ENFP-tree and FP-array ,and mine ENFP-tree, the constructed condition FP-tree needs other space to be store. And implement until to the end of the recursive algorithm.

2) ENFP-growth algorithm advantage analysis

To compare with FP-growth, ENFP-growth has algorithm efficiency as follows:

①Compress the data with node switching strategy, reduce the space of storing data;

②Get the support number of frequent item with FP-array directly, thus omit to the first traversal of the FP-tree.

③Because of FP-array has saved the count of each frequent item, then the count of item header table in ENFP-tree could be obtained from FP-array, so it do not need get the accumulation count of item from item link table.

④It avoids dividing other space to store condition FP-tree with handling head table from down to up and item link from backward. Though there is not every item has the condition of switching in ENFP-growth algorithm, but there has some item with the switching condition to compress the data. In ENFP-growth algorithm, not every item could store condition FP-tree in primate tree.

But part of condition FP-tree needs no more space by such algorithm proceeding. ENFP-growth algorithm needs to optimize in two aspects of node switching and item proceeding backward. It needs to accord to the actual situation of data. On one hand, it could not switch nodes and impact the item backward mapping. On the other hand, it could not pursue reducing the space of storing the condition FP-tree in terms of items could proceed backward and reduce the nodes with switching condition. So when ENFP-growth algorithm is constructing ENFP-tree, it switches the items behind the item header table, the items in front it does not need to be switched. When proceeds the items backward, if there is no items behind this item, it could map the item backward.

To a tree, mine the nodes behind item header table, the depth of the condition and the involved nodes are certain. So it does not need more other space to store the condition FP-tree generated by the item. On the contrary, it will be large about the degree of the condition FP-tree of the item in front of the item header table ,and could involve more nodes, so if it could project and store condition FP-tree in the primate FP-tree, more space will be saved. So the algorithm strengthens the compactness of the tree in a certain extent and improves the space utilization.

3) ENFP-growth algorithm example

It shows the implementation process with an example as follows: the transaction database is showed as Table 1, the minimum support threshold is 2. After it scans the primate database at first time, then order the frequent itemsets as decreasing: {I6:4,I3:4,I1:3,I2:3,I8:3,I10:3}, and store the item to Table L as shows in Table 2:

TABLE I. INSTANCE DB

TID	The original item set	after finish the project set
T1	I6, I1, I3 , I4, I7, I9, I8, I10	I6, I3, I1, I8, I10
T2	I1, I2, I3, I6 , I12, I8, I15	I6, I3, I1, I2, I8
T3	I2, I6, I13, I16, I15	I6, I2
T4	I2, I3, I11, I17, I10	I3, I2, I10
T5	I1, I6, I3, I5, I12, I10, I8, I14	I6, I3, I1, I8, I10

TABLE II. ITEM HEADER TABLE L

item	Support degree
I6	4
I3	4
I1	3
I2	3
I8	3
I10	3

Construct ENFP-tree, FP-array according to the algorithm 1 with the data in table I, II ,and are as shown in figure I, II.

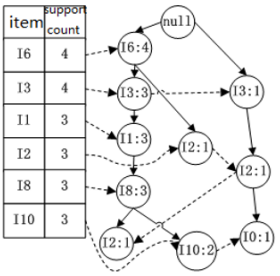


FIGURE I. ENFP-TREE

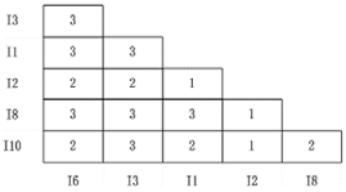


FIGURE II. FP-ARRAY

Step one: Calculated the frequent itemsets constitute by the frequent item I_{10} and the back of the item (no) of I_{10}. Figure III illustrates there are no item behind I_{10}, Untreated item behind I_{10} without having to open up space to store condition ENFP-tree. Support number of I_{10} is 3 from the header table L, so the first step for the frequent itemsets: $\{I_{10}:3\}$.

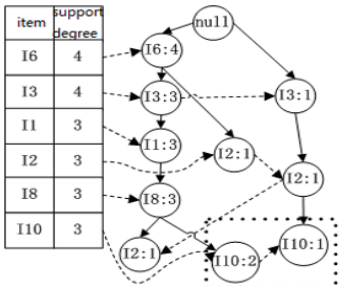


FIGURE III. BACKWARD MAPPING OF I_{10}

Step 2: Calculated the frequent itemsets with same procedure, then the frequent itemsets in the second step is $\{I_8:3\}$.

Step 3: Calculated the frequent itemsets with same procedure, then the frequent itemsets in the third step is $\{I_2:3\}$.

Step 4: Calculated the frequent itemsets with same procedure,

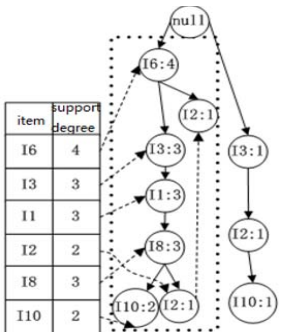
So the frequent items in the fourth step are $\{\{I_1:3\}, \{I_1I_8:3\}\}$.

Step 5: Calculated the frequent itemsets with same procedure

So the frequent itemsets obtained in the fifth step are $\{\{I_3:4\}, \{I_3I_{10}:3\}, \{I_3I_8:3\}, \{I_3I_1:3\}, \{I_3I_1I_8:3\}\}$.

Step 6: When constitute condition ENFP-tree of I_6 and FP-array at the same time, the obtained FP-array is as shown in figure 4(a). Calculated the frequent itemsets constituted by the frequent item I_6 and the items (I_{10}, I_8, I_2, I_1, I_3) behind of I_6. Support number of I_6 is 4 from the header table L. There exist

items I_{10}, I_8, I_2, I_1, I_3 behind I_6 from figure IV(a). The support number of I_{10}, I_8, I_2, I_1 and I_3 are respectively 2, 3, 2, 3, 3, when I_6 exists by the FP-array from figure II. So there exists I_6I_8:3, I_6I_1:3, I_6I_3:3. There is no untreated items behind I_6, so it needs no new space to store condition ENFP-tree. There is no untreated items behind I_{10}, it needs no consideration. For there exists items behinds I_8, I_8 needs to be considered. The items behind I_8 are non-frequent items from figure IV (b). I_8 is the frequent item behind I_1 from figure IV (b). So I_6I_1 could be extends as $I_6I_1I_8$:3; When constitute I_6I_3 conditions ENFP-tree and constitute FP-array at the same time, it gets the FP-array as shown in figure IV (c). It shows the frequent item behind I_3 is I_8 and I_1. So I_6I_3 could be extended as $I_6I_3I_8$:3, $I_6I_3I_1$:3. There is no untreated items behind I8 from IV (c), so it needs no extension. It shows the frequent item behind I_1 is I_8 in figure IV (c). So $I_6I_3I_1$could be extended as $I_6I_3I_1I_8$:3. Then the obtained frequent itemsets in the sixth step are:{{I_6:4},{I_6I_8:3},{I_6I_1:3},{I_6I_3:3},{$I_6I_1I_8$:3},{ $I_6I_3I_8$:3},{ $I_6I_3I_1$:3},{ $I_6I_3I_1I_8$:3}}.



(a) Backward mapping of I_6

(b) The FP-array of I_6 (c) the FP-array of I_6I_3

FIGURE IV. (a), (b), (c)

At last, we obtain all of the frequent itemsets: {I_{10}:3},{I_8:3},{I_2:3},{I_1:3},{I_1I_8:3},{I_3:4},{I_3I_{10}:3},{I_3I_8:3},{I_3I_1:3},{$I_3I_1I_8$:3},{I_6:4},{I_6I_8:3},{I_6I_1:3},{I_6I_3:3},{$I_6I_1I_8$:3},{$I_6I_3I_8$:3},{$I_6I_3I_1$:3},{$I_6I_3I_1I_8$:3} by all steps above.

E. Contrast Experiment

In order to verify the validity of the ENFP - growth algorithm,

It makes a contrast with ENFP-growth algorithm, FP-growth algorithm TFP-growth based on two-dimensional table, classic FP-growth algorithm in terms of memory use and run time in this article .The data adopted in this article is T20.I6.450K dataset, the dataset includes 450000 transactions. There are 20 items of each transaction on the average. The experiment is in the hardware environment for Intel Core2 Duo CPU, 2.0 G of memory, WIN7 operating System. Experimental procedure is implemented by java, and the experiment environment is in IDEA compiling environment.

Compare memory usage of three algorithms and show in Figure V:

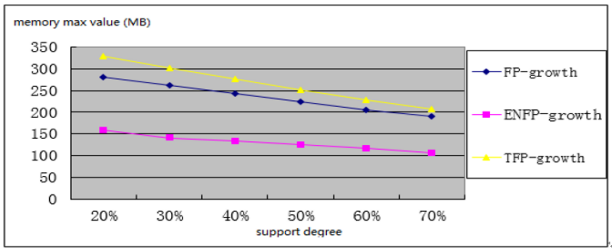


FIGURE V. COMPARISON OF THREE ALGORITHMS IN MEMORY USAGE

ENFP-growth algorithm takes the node switching strategy and compress the transaction form compact tree; it could store the information of condition FP-tree in the primate tree with the backward mapping. All above method could decrease memory usage, so ENFP-growth algorithm has high efficiency than FP-growth algorithm in memory usage. Though ENFP-growth algorithm needs additional space to construct FP-array. But ENFP-growth algorithm has much less memory usage max value than FP-growth algorithm in whole running result of algorithm. When the support degree is less, but the item of transaction is many, and many items are to be compressed and generated great condition FP-tree. So FP-growth algorithm needs additional space to store FP-array, but ENFP-growth algorithm needs no additional space. When the support degree is less, ENFP-growth algorithm has much less memory usage max value than FP-growth algorithm. But with the support degree increasing, there are less nodes could be switched nodes, so condition FP-tree could be small. And the difference between ENFP-growth algorithm memory max value and FP-growth algorithm memory max value is less. With the support degree increasing, ENFP-growth algorithm save FP-array all the time, so the decreasing of ENFP-growth algorithm memory max value is slow than FP-growth algorithm. TFP-growth algorithm builds the relation of each pair of the items with two dimensional table, so the memory occupancy rate of FP-growth algorithm is higher than ENFP-growth algorithm in Figure VI.

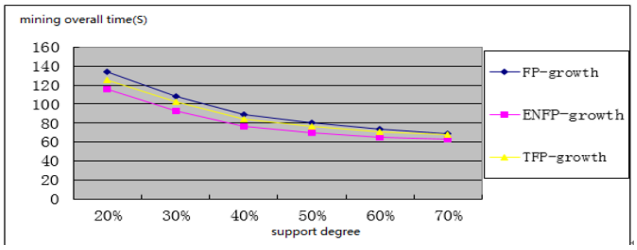


FIGURE VI. THE RUNNING TIME OF THE THREE ALGORITHMS IS COMPARED

F. Summary and Scope

This paper mainly studies the data mining of association rules algorithm. The article starts with the classical algorithm of association rules of FP-growth algorithm. By carefully analyzing the FP-growth algorithm, it finds that FP-growth algorithm has the defect of not thorough data compression, more searching time of FP-tree, more space to store condition FP-tree. To the defect of FP-growth algorithm, it puts with the improvement algorithm of FP-growth algorithm in this paper, combines node switching strategy, FP-array auxiliary structure

with backward mapping. And generates a new FP-growth algorithm-ENFP-growth algorithm. Then implements ENFP-growth algorithm. And it makes a contrast with ENFP-growth algorithm, TFP-growth algorithm, classical FP-growth algorithm. The experimental result shows ENFP-growth algorithms prior to FP-growth algorithm and TFP-growth algorithm. In order to solve the problem of dealing with huge amounts of data, implements the ENFP-growth algorithm in Spark environment, the experiment shows ENFP-growth algorithm has less time consuming when it is running on the enormous data and more than two nodes than FP-growth algorithm on single-machine environment.

This article proposes an improvement to classic FP-growth algorithm and improves the time efficiency and space efficiency. But FP-growth ALG exists following problems:

1) ENFP-growth algorithm constructs compact tree with node switching strategy, it improves the compressing efficiency of data. But in ENFP-growth algorithm, not all the nodes satisfy the switching strategy, so the compressing efficiency needs to be improved further.

2) ENFP-growth could store the condition FP-tree of the part of items in the primate tree with backward mapping. It improves the utility efficiency of memory. But not all the condition FP-tree of each item could be stored in the primate tree with ENFP-growth algorithm. So it needs to study more efficient methods to solve the problems of memory utilization further.

3) This article studies the method of mining frequent itemsets in the condition of static database, but the data is changing all the time, so it needs to study method of mining frequent itemsets of updating real time database.

ACKNOWLEDGEMENTS

This work was supported by the industrial Public Relation Project of Shaanxi Technology Committee [2016GY-140] and Natural Foundation of Shaanxi Province [2016JM5091]

REFERENCE

[1] C. Borgelt. Recursion Pruning for the Apriori Algorithm. Proc. 2nd IEEE ICDM Workshop on Frequent Item Set Mining Implementations (FIMI 2003, Brighton, United Kingdom). CEUR Workshop Proceedings 126, Aachen, Germany 2004.

[2] Christian Borgelt *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 2(6):437-456.
J. Wiley & Sons, Chichester, United Kingdom 2012

[3] Keeping Things Simple: Finding Frequent Item Sets by Recursive Elimination Christian Borgelt. *Workshop Open Source Data Mining Software (OSDM'05, Chicago, IL)*, 66-70. ACM Press, New York, NY, USA 2005

[4] Han, J., Pei, J., Yin, Y .: Mining Frequent Patterns without Candidate Generation, ACM SIGMOD 2000, Dallas, TX, U. S. A.

[5] Wang Yifei , Zhang Yong Nanjing University of Aeronautics and Astronautics, Information Science and technology college, Nanjing 210016;A Algorithm for Mining Maximal Frequent ItemSets Based on Condition Pattern [A];[C];2005

[6] Zhao Qun-li(Department of Computer Science and Technology, Anhui Institute of Education,Hefei,230061);An Integrated Updating Algorithm for Mining Maximal Frequent Itemsets Based on FP-Tree[J];Journal of Anhui Institute of Education;2006-03

[7] University,Fuzhou 350002,China);Algorithm for Mining Frequent Patterns Based on Merged FP-tree[J];Journal of Guangxi Normal University(Natural Science Edition);2007-04

[8] Auxiliary matrix formalism for interaction representation transformations, optimal control, and spin relaxation theories J Chem Phys. 2015 Aug 28; 143(8):084113. doi: 10.1063/1.4928978

[9] Yun Yang, Yanxia Luo. The improvement of FP-growth algorithm [J]. Computer engineering and design 2010，31（7）：1506-1509.