International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 1 Issue 10, December- 2012

Use Of Different Data Structures In Association Rule Mining: A Survey

Adhav Ravi
M.Tech-II
SGGS IE&T,Nanded.

Bainwad A. M.
Assistant Professor
SGGS IE&T, Nanded.

Abstract

Association rule mining is important data mining task for which many algorithms have been proposed. All these algorithms generally work in two phases, finding frequent itemsets and generating association rules from them. First phase is most time consuming in most of the algorithms because algorithm has to scan the database many times. Use of different data structures overcomes this drawback. In this paper we will survey the algorithms which make use of different data structures to improve association rule mining

1. Introduction

Association rule mining, one of the most important and well researched techniques of data mining, was first introduced in [1]. It aims to extract interesting correlations, frequent patterns, associations or casual structures among sets of items in the transaction databases or other data repositories. Association rules are widely used in various areas such as telecommunication networks, market and risk management, inventory control etc. Various association mining techniques and algorithms will be briefly introduced later.

Association rule mining is to find out association rules that satisfy the predefined minimum support and confidence from a given database. The problem is usually decomposed into two sub problems. One is to find those itemsets whose occurrences exceed a predefined threshold in the database; those itemsets are called frequent or large itemsets. The second problem is to generate association rules from those large itemsets with the constraints of minimal confidence. Suppose one of the large item-sets is L_k, $L_k = \{I_1, I_2, \dots, I_k\}$, association rules with this itemsets are generated in the following way: the first rule is $\{I_1, I_2, \dots, I_{k-1}\} = \{I_k\}$, by checking the confidence this rule can be determined as interesting or not. Then other rule are generated by deleting the last items in the antecedent and inserting it to the consequent, further the confidences of the new rules are checked to determine the interestingness of them. Those processes iterated until the antecedent becomes empty. Since the second sub-problem is quite straight forward, most of the researches focus on the first sub-problem.

The first sub-problem can be further divided into two sub-problems: candidate large item sets generation process and frequent itemsets generation process. We call those item sets whose support exceed the support threshold as large or frequent itemsets, those itemsets that are expected or have the hope to be large or frequent are called candidate itemsets.

In many cases, the algorithm needs to scan data base for number of times to generate frequent itemsets which causes inefficiency of algorithm. Several strategies have been proposed to reduce time complexity of algorithm. One of these strategies is to use different data structures based algorithms for finding frequent item sets such as tree, graph and matrix.

2 Use of different data structures in association rule mining

2.1 Graph

2.1.1 PAPG (Primitive Association Pattern Generation)

In this algorithm [2] the first step is to construct association graph. This is two-step process numbering and graph construction. In the numbering phase, the algorithm PAPG arbitrarily assigns each item a unique integer number. In the large item generation phase, PAPG scans the database and builds a bit vector for each item. The length of each bit vector is the number of transactions in the database. If an item appears in the ith transaction, the ith bit of the bit vector associated with this item is set to 1. Otherwise, the ith bit of the bit vector is set to 0. The bit vector associated with item i is denoted as BVi. The number of 1s in BVi is equal to the number of transactions which support the item i, that is, the support for the item i. For association graph construction PAPG uses AGC (Association Graph Construction) algorithm. The AGC algorithm is described as follows: For every two large items i and j($i < j$), if the number of 1s in $BV_i \Lambda BV_j$ achieves the user-specified minimum support, a directed edge from item i to item j is created. Also, itemset (i, j) is a large 2-itemset.

Second step is to generate Primitive Association Pattern. The large 2-itemsets are generated after the association graph construction phase. In the association pattern generation phase, the algorithm LGDE (Large itemset Generation by Direct Extension) is proposed to generate large k–itemsets (k > 2), which is described as follows: For each large k-itemset($k \geq 2$), the last item of the k-itemset is used to extend the large itemset into k+1-itemsets.

Suppose $(I_1, I_2, \dots, I_k)$ is a large k-itemset. If there is no directed edge from item Ik to an item v, then the itemset need not be extended into k+1-itemset,because $(I_1, I_2, \dots, I_k, v)$ must not be a large itemset. If there is a directed edge from item Ik to an item u, then the itemset $(I_1, I_2, \dots, I_k)$ is extended into$K + 1 - \text{itemset}(I_1, I_2, \dots, I_k)$. The itemset $(I_1, I_2, \dots, I_k, u)$ is a large $k + 1 - \text{itemset}$ if the number of 1s in $BV_1 \Lambda BV_2 \Lambda \dots \Lambda BV_{ik} \Lambda BV_u$ achieves the

minimum support. If no large k+1-itemsets can be generated, the algorithm LGDE terminates.

2.1.2 GAPG (Generalized Association PatternGeneration)

GAPG [2] is used to discover all generalized association patterns. To generate generalized association patterns, one can add all ancestors of each item in a transaction to the transaction and then apply the algorithm PAPG on the extended transactions.

In the numbering phase, GAPG applies the numbering method PON (POstorder numbering method) to number items at the concept hierarchies. For each concept hierarchy, PON numbers each item according to the following order: For each item at the concept hierarchy, after all descendants of the item are numbered, PON numbers this item immediately, and all items are numbered increasingly. After all items at a concept hierarchy are numbered, PON numbers items at another concept hierarchy.

In the large item generation phase, GAPG builds a bit vector for each database item, and finds all large items (include database items and generalized items). Here, we assume that all database items are specific items.

In the association graph construction phase, GAPG applies the algorithm GAGC (Generalized Association Graph Construction) to construct a generalized association graph to be traversed. The algorithm GAGC is described as follows: For every two large items i and j ($i < j$), if item j is not an ancestor of item i and the number of 1s in $BV_i \Lambda BV_j$ achieves the user-specified minimum support, a directed edge from item i to item j is created. Also, itemset (i, j) is a large 2-itemset.

In the association pattern generation phase, GAPG applies the LGDE algorithm to generate all generalized association patterns by traversing the generalized association graph.

2.1.3 Undirected Item Set Graph [3]

Undirected item set graph is set of nodes $V(V_1, V_2, \ldots, V_n)$ in the database. Each node contains: the node name, the pointer to other nodes, and the number of nodes to which it points. The side set $E < i, j >$ of undirected item set graph has two attributes: the side name and the number of side appear. $< V_i, V_j >$ Express two frequent itemsets;$< V_1, V_2, \ldots, V_n >$express n frequent itemset.

In construction of Undirected Item Set Graph First step is to scan the database. It makes each item as a node and at the same time it makes the supporting trade list for each node. Supporting trade list is a binary group $T = \{T_{id}, \text{Itemset}\}$ (where T_{id} is transaction id and Itemset is trade item set). So the side between nodes can be accomplished by corresponding trade list operation. The algorithm does the intersection of two nodes with supporting trade list. When trade list is not empty, that means there is a side between two nodes. The appearance number of each side is the resultant number which algorithm finds by the side's intersection.

Algorithm one: Construction of undirected item sets graph

Input: Database D

Output: Undirected item set graph

Begin

1. Add the items into the vertex set V;
2. For $i = 1$ to $n - 1$
   2.1. Select V_i fromV;
   2.2. For each V_j ($j \neq i$)
      2.2.1. If $(I_i \cap I_j) \neq \emptyset$ then
   2.2.2. Add link between V_i and V_j//V_i and V_j Become adjacent nodes.
   2.2.3. End if.
   2.3. Next
3. Next

End

The algorithm in [3] uses the search strategy of Depth first-Search to set universal undirected item graph. The specific steps are shown as follows: Select a node V_i from node set V. If the number of times V_i appears in the database is not less than the minimum support minsupp, then $\{V_i\}$ will belong to the item in frequent 1-item set. If count of node V_i adjacent to node V_j's side is not less than support S, then $\{V_i, V_j\}$ will belong to the item in frequent 2-itemset. When there are three nodes in undirected item set graph and count of each side of the node is not less than minimum support minsupp, these three nodes $< V_k, V_m, V_n >$ will belong to frequent 3-item set. When there more than three nodes in undirected item sets graph then count of each side of the node should not be less than minimum support minsupp and all the subset of these n nodes should be frequent.

Algorithm two: To find frequent item set based on undirected item sets graph.

Input: Undirected item set graph, minimum support minsupp, minconf

Output: frequent item set L, Association rule

Begin

1. The node set V is empty or not. If it is empty then stop;
2. Find count of each item (e.g.V_i) and check count of each item is greater than or equal to minimum support minsupp. If greater than the items are stored in frequent-1 item set;
3.(frequent item set) = L;
4. Select any unvisited node (e.g.V_j) from adjacent list OfV_i;
5. If count $((V_i, V_j) \geq$ minsuppp) then
   5.1. L U V_j;
   5.2. L. adjacentlist = (L. adjacentlist) $\cap$ (V_j . adjacent list);
   5.3. Call DFS (V_j) Procedure;
6. End if;
7. Confidence of each item is compared with minconfand strong association rules are generated.
8. End;

ProcedureDFS (V_j):

Begin

1. If V_j. adjacentlist $\neq \Phi$ then
   1.1. Select any other node, suppose $V_k$from

V_j. adjacentlist;
1.2. Call isloop (L, V_k) Procedure;
1.3. If count (L, V_k) is greater than or equal to minimum support then combine L U (V_k).
1.3.1. Call DFS (V_k);
1.3.2. Output is frequent item set;
1.3.3. Delete V_k fromV_j. adjacentlist;
1.3.4. Call DFS (V_j);
1.4. Else Return to its parent vertexV_i;
1.5. Call DFS (V_i);
2. End;

Procedureisloop (L, V_k):
Begin
1.If V_k € L. adjacentlist then return V_k;
2. Else delete V_k fromV_j. adjacentlist;
3. CallDFS (V_j);
4. End;

When database and minimum support i.e. minsupp is changed the undirected graph should be changed accordingly. If we want to add some new items to the database, then undirected item set graph is updated accordingly. At this time, the new frequent item sets can be found only by running algorithm two again. When the minimum support is changed, new frequent item set can be found only by adjusting the parameter of algorithm two again.

2.1.4 DLG

DLG [4] is a three-phase algorithm. The large 1-itemset generation phase finds large items and records related information. The graph construction phase constructs an association graph between large items, and at the same time generates large 2-itemsets. The large item set generation phase generates large k-itemsets $(k > 2)$ based on this association graph.

In large 1-itemset generation phase, the DLG algorithm scans the database to count the support and builds a bit vector for each item. The length of a bit vector is the number of transactions in the database. The bit vector associated with item i is denoted asBV_i. The j th bit of BV_i is set to 1 if item i appears in the j th transaction. Otherwise, the j th bit of BV_i is set to 0. The number of 1's in BV_i is equal to the support count of the item.

In graph construction phase, the support count for the itemset $\{I_1, I_2, \dots, I_k\}$ is the number of 1's in$BV_{i1} \Lambda BV_{i2} \Lambda \dots \Lambda BV_{ik}$, where the notation "$\Lambda$" is a logical AND operation. Hence, the support count of the itemset $\{I_1, I_2, \dots, I_k\}$ can be found directly by applying logical AND operations on the bit vectors of the k-itemsets instead of scanning the database. If the number of 1's in$BV_i \Lambda BV_j (i < j)$ is no less than the minimum support count, a directed edge from item i to item j is constructed in the association graph. Also, $\{i, j\}$ is a large 2-itemset.

In large itemset generation phase, for each large k-itemset$\{I_1, I_2, \dots, I_k\}$ inL_k $(k > 1)$, the last item i_k is used to extend the itemset into $(k+1)$-itemsets. If there is a directed edge from i_k to itemj, the itemset $\{I_1, I_2, \dots, I_{kj}\}$ is a candidate $(k+1)$ -itemset. If the number of 1's in $BV_{i1} \Lambda BV_{i2} \Lambda \dots \Lambda BV_{ik} \Lambda BV_j$ is no less than the minimum support count, $\{I_1, I_2, \dots, I_{kj}\}$ is a large $(k+1)$-itemset in L_{k+1}. If no large k-itemset is generated in the kth iteration, the algorithm terminates.

2.1.5 DLG*

In the kth $(k > 2)$ iteration, DLG [4] generates candidate k -itemsets by extending each large $(k-1)$ -itemset according to the association graph. Suppose on the average, the out-degree of each node in the association graph is q. The number of candidate itemsets is $|L_k - 1| \times q$, and DLG must perform $|L_k - 1| \times q \times (k-1)$ logical AND operations on bit vectors to determine all large k-itemsets. The key issue of the DLG* [4] algorithm is to reduce the number of candidate itemsets.

In the large itemset generation phase, DLG* extends each large k-itemset in $L_k (k \geq 2)$ into $(k+1)$-itemsets like the original DLG algorithm. Suppose $\{I_1, I_2, \dots, I_k\}$ is a large k-itemset, and there is a directed edge from item i_k to item i. If the $(k+1)$-itemset $\{I_1, I_2, \dots, I_k, I\}$ is large, it must satisfy the following two conditions (Otherwise, it cannot be large and is excluded from the set of candidate $(k+1)$-itemsets).
1. Any $\{i_j, i\}$ $(1 \leq j \leq k)$ must be large. In other words, the in-degree of the node associated with item i must be at least k.
2. Moreover, a directed edge from i_k to item i means that $\{i_k, i\}$ is also a large 2-itemset. Therefore, we only need to check if all $\{i_j, i\}(1 \leq j \leq k-1)$ are large.

These simple checks significantly reduce the number of candidate itemsets. In order to speed up these checks, we record some information during the graph construction phase. For the first condition, for each large item, we count the in-degrees of this item. For the second condition, a bitmap with $|L_1| \times |L_1|$ bits is built to record related information about the association graph. If there is a directed edge from item i to item j, the bit associated with $\{i, j\}$ is set to 1. Otherwise, the bit is set to 0. DLG* requires extra memory space of size quadratic to |I|, but speeds up the performance significantly.

2.2 Matrix

2.2.1 ABBM

In general, the ABBM algorithm [5] consists of four phases as follows:
1. Transforming the transaction database into the Boolean matrix
2. Generating the set of frequent 1-itemsets L1
3. Pruning the Boolean matrix
4. Generating the set of frequent k-itemsets $L_k (k > 1)$

In the first step the mined transaction database is D, with D having m transactions and n items. Let T={T_1,T_2,..,T_m} be the set of transactions and I={I_1,I_2,..I_n}be the set of items. We set up a Boolean matrix A m*n, which has m rows and n columns. Scanning the transaction database D, if item Ij is intransaction T_i, where$1 \leq j \leq n, 1 \leq i \leq m$, the element value of A_{ij} is '1,' otherwise the value of A_{ij} is '0'.

In the second step, The Boolean matrix A m*n is scanned and support numbers of all items are computed. The support number I_j.supth of item I_j is the number of '1s' in the jth column of the Boolean matrix A m*n. If I_j.supth is smaller than the minimum support number minsupth, itemset $\{I_j\}$ is not a frequent 1-itemset and the j^{th} column of the Boolean matrix A m*n will be deleted from A m*n. Otherwise itemset $\{I_j\}$ is the frequent 1-itemset and is added to the set of frequent 1-itemset L_1.

Pruning the Boolean matrix means deleting some rows and columns from it. First, the column of the Boolean matrix is pruned according to Proposition 2. This is described in detail as: Let I′ be the set of all items in the frequent set L_{K-1}, where k>2. Compute all $|L_{K-1}(j)|$ where I′, and delete the column of correspondence item j if $|L_{K-1}(j)|$ is smaller than k-1. Second, recompute the sum of the element values in each row in the Boolean matrix.

Frequent k-itemsets are discovered only by "and" relational calculus, which is carried out for the k-vectors combination. If the Boolean matrix A p*q has q columns where $2<q\leq n$ and $minsup^{th}\leq p\leq m$, C_q^k , combinations of k-vectors will be produced. The 'and' relational calculus is for each combination of k-vectors. If the sum of element values in the "and" calculation result is not smaller than the minimum support number minsupth, the k-itemsets corresponding to this combination of k-vectors are the frequent k-itemsets and are added to the set of frequent k-itemsets L_k.

2.2.2 TCOM

In order to employee the advantages of both horizontal and vertical layouts,[6] uses matrix structure called Transactional Co-Occurrence Matrix, in short TCOM. The algorithms designed on the base of TCOM are very efficient and fast after it is constructed since full access of original database or TCOM is no longer necessary.

A Transactional Co-Occurrence Matrix is an innovative variant of a co-occurrence matrix [7]. A co-occurrence matrix is a square two dimensional matrix, whose rows and columns are items, or called attributes. If there are M items (attributes) in the database, the size of the corresponding co-occurrence matrix will be M*M.

It is easy to notice that a co-occurrence matrix is great to mine the simple rules but is impossible to mine a high-degree rule since the transactional information of 3 or more items are lost during the construction of the matrix. But such rules are desired for most of the time. Another drawback of the co-occurrence is that the items are not sorted according to their occurrence counts, which will significantly slow down the item set searching during the mining process.

To overcome the above short comings, algorithm incorporates transactional information into a sorted co-occurrence matrix and makes it suitable for all association rule-mining tasks.

The transform from the original database into the transactional co-occurrence matrix layout requires two passes of the database. The first pass of the original database is to count the occurrence of each item and sort items into descending order according to their occurrence counts. During the second pass of the original database, each transaction is sorted and then inserted into the transactional co-occurrence matrix.

TCOM has great advantage by combining the transactional oriented information with item oriented information in to one single structure. During the mining process, two pieces of information are needed.
1. For a given transaction, we need to know what items it contains;
2. For a given item set, we need to know the occurrence count of this item set.

If we only use the horizontal layout database (the original database) to do the mining problem, then a full access of the database is needed every time when the occurrence count of an item set is desired. On the other hand, if we only use the vertical layout database then a full access of the database is needed every time when the first kind of information is desired.

Mining process

Unlike previous literature which has to find all itemsets before finding the valid association rule, we directly find the valid association rules and itemsets simultaneously. We call our mining process as TCOM_mining. It is an item oriented algorithm and the simplified version is shown below.

TCOM_mining:
1.Let set I be the set of infrequent items, $I=\{i_1;i_2,\ldots\ldots,i_n\}$
//the items in I is in decreasing order according to their occurrence count, such as
//$occurrence\_count(i_1)>=occurrence\_count(i_2)>=\ldots.>=occurrence\_count(i_n)$,
//which can be obtained directly from the TCOM
2. Start with item in, for each item ir in the set I, 1<=r<=n
2.1 Let ISSET be the set of itemsets, initially ISSET is an empty set
2.2 Find out all existing item set ISA = $\{is_1; is_2,\ldots.,i_r\}$ where $occurrence\_count(is_1) >= occurrence\_count(is_2) >= \ldots. >= occurrence\_count(i_r)$
2.3 Populate ISSET with itemsets found in step 2.2
2.4 Find out occurrence count for each ISA found in step2.2
3. For each item set ISA in the set ISSET
//find two kinds of rule: the rules in which i_r is in the antecedent and i_r is the least
//frequent item and the rules in which i_r is in the antecedent and i_r is only the least
//frequent item in the antecedent but not in the whole rule
//step 3.1 is to find first kind of rules
3.1 For each item set ISB in the set ISSET where ISB != ISA
3.1.1 If ISB contains ISA
3.1.1.1 Let ISC be the difference of ISB and ISA
3.1.1.2 If occurrence_count (ISB) ≥occurrence_count (ISA)*σ
// σ is the minimal confidence threshold
3.1.1.2.1 ISA -> ISC is a valid rule
End if

//occurrence_count (ISB) >occurrence_count (ISA)*σ End if //ISB contains ISA
End for
//each item set ISB in the set ISSET
//steps 3.2-3.4 are to find second kind of rules
3.2 Find out all happened item set ISB where ISB contains ISA, and there exist at least one item j in ISB with occurrence_count(j) <occurrence_count(i_r)
3.3Find out the occurrence count for each ISB found in step 3.2
3.4For each itemset ISB found in step 3.2
3.4.1 Let ISC be the difference of ISB and ISA
3.4.2 If occurrence_count (ISB) ≥ occurrence_count (ISA)* σ
3.4.2.1 ISA ->ISC is a valid rule
End if
// occurrence_count(ISB) >= occurrence_count(ISA)
End for// each item set ISB found in step 3.2
End for// each item set ISA in the set ISSET

2.2.3 Algorithm BitMatrix

In Apriori and AprioriTid algorithms, it is assumed that items in each transaction are kept sorted in their lexicographic order [8]. However, this is not needed in BitMatrix. By careful programming, we can keep the items in the large itemsets and the large itemsets of the same size are kept sorted in their lexicographic order even if the items in the transactions are not kept sorted. We call the number of items in an item set its size, and call an item set of size k a k-item set. The set of all large k-itemsets is defined as L_k. Each k-item set c in Lk consists of items c[1],c[2],...,c[k], where c[1] < c[2] <…< c[k]. Associated with each item set are two fields: count field to store the support for this itemset, and index field (henceforth referred to as support index) to indicate the transactions that contain the itemset. The BitMatrix algorithm is described as:

(1) Initialize the bitmatrix;
(2) L_1 = {large 1-itemset};
(3) for (k=2; L_k!=0; k++) do
(4) L_k =GenLargeItemsets(L_{k-1});
(5) Answer= $U_k L_k$.

In Step (1) of this algorithm, we initialize the bitmatrix as follows. First we build a matrix whose row number and column number are the item number and the transaction number, respectively. Note that the matrix is a bit-matrix and every position of the matrix only has one bit in the memory. Then we go through the database. If there are items i_1, i_2,...,i_k in the jth transaction, bits a_{i1j}, a_{i2j}, ... , a_{ikj} (a_{ij} represents the bit of i^{th} row and j^{th} column) and the other bits in the j^{th} column of the matrix are initialized as 1 and 0 respectively.

In Step (2), we simply count the number of 1 in each row to get the support count of every item and the large 1-itemsets axe determined.

In Step (4), the previously generated large (k-1)-itemsets are used to generate the large k-itemsets. This step repeats until no new large itemsets are generated. The GenLaxgeItemsets function is used here, which takes as argument L_{k-1} and returns L_k. The function works as follows.

(1) for($\forall p, q \in L_{k-1}$) do
(2)if$p[1] = q[1] \Lambda, \dots, \Lambda(p[k-2] = q[k-2] \Lambda (p[k-1] < q[k-1])$ *then* {
(3) c = p ∪ q; //c consists of p[1], p[2] ..... p[k - 2], p[k - 1], q[k - 1]
(4) for all (k - 1)-subsets s of c do
(5) if(s ∉ L_k) then {delete c; c = 0; break;}
(6) if(c ≠ 0) then {
(7) c.index = p.index&q.index; //support index
(8) computec.count from c.index; //support count
(9) if (c.count>minsup) then Lk = Lk U {c};
(10) }//end if
(11) }//end if

From Steps (1) to (5), the function simply helps generate the Ck that is a set of candidate k-itemsets (potentially large itemsets, see also [8]). In Step (2), the condition p[k - 1] <q[k - 1] ensures that no duplicates are generated. However, this algorithm differs from Apriori in that it need not store all the candidates in the memory. Once a candidate itemset is generated, it will be determined in Steps (7) to (9) whether it is a large one.

To decide whether a candidate item set is a large one, we associate each large itemset with a support index, which is a bit index and each bit of which indicates whether the itemset is contained by a transaction in the database. As to the 1-itemsets, their support index is some row in the bitmatrix.

Since c is the union of p and q, we simply generate c's support index by bit operator AND ("&") that is applied to each bit of p's and q's in Step (7).

2.3 Tree

2.3.1 TBAR

TBAR [9] is aApriori based association rule mining which uses tree data structure to store relevant itemsets in database. Use of itemset tree to store relevant itemsets saves space and time required to process data. TBAR was mainly developed to work with relational databases. It makes each item as pair column_name:value. It will use the following algorithm to find all the relevant itemsets:

```
set.Init (MinSupport);
itemsets =  set.Relevants(1);
k = 2;
while (k <= columns && itemsets >= k) {
    itemsets = set.Candidates(k);
        If (itemsets>0)
      Itemsets = set.Relevants(k);
    k++;
}
```

In this algorithm the set is itemset tree. init method will initialize the itemset tree. Method relevants(k) will generate L_k and candidate(k) will generate Ck from L_{k-1}.

The itemset tree will look like

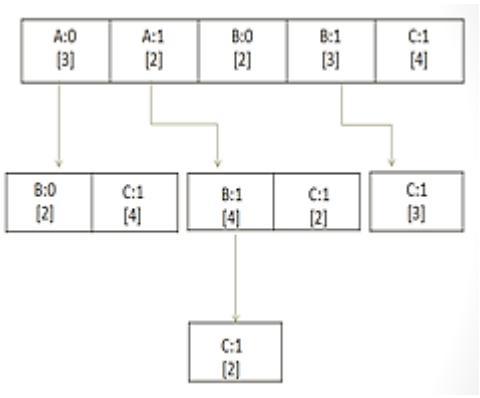

Fig.1 TBAR data structure example.

2.3.2 STBAR

STBAR [10] is extended version of TBAR. STBAR employs the tree-based storage, which is analogous to TBAR. Each node in the tree is not a 2-tuple <a, v>, but a 3-tuple <a, v, t>, in which 'a' is the attribute, 'v' is the number of tuples which satisfy the condition, 't' is a flag whose value is 1 or 0. This flag will decide whether an item can concatenate the items found in the paths from the root of the tree to the current i tem or not.

Step1: Generate all the frequent 1-itemsets, and store them in the 3-tuples.

Step2: According to the order $I_{n-1} \ldots I_1$, generate itemset L_2, validate whether it can constitute a frequent itemsets or not, and set the corresponding flag in the 3-tuple.

Step3: For each sub-itemset in L_2, recursively generate L_n according to Step 2.

Step4: Seek for the tree's depth.

Step5: Find out the longest concatenations in the tree.

Step6: Produce all the association rules. This step is just as TBAR doing.

The STBAR datastructure will look like

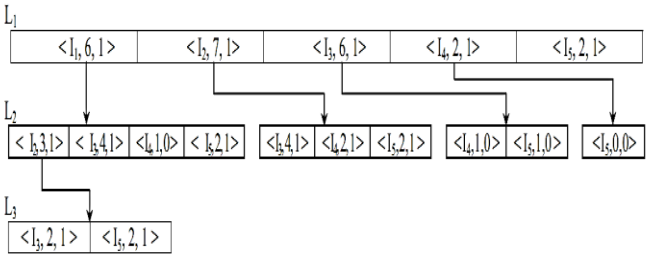

Fig.2STBAR data structure example.

3. Conclusion

Association rule mining is widely used in market basket analysis, medical diagnosis, Website navigation analysis, homeland security and so on. During association rule mining most of the time is spent for scanning database for finding frequent itemsets. This time can be reduced by using different data structures to store frequent itemsets. In this paper we surveyed the mining algorithms which make use of different data structures to reduce space and time complexity of algorithms.

4. References

[1] Agrawal R., Imielinski T. and Swami A. N., "Mining Association Rules Between Sets of Items in Large Databases," ACM SIGMOD International Conference on Management of Data, pp. 207-216, 1993.

[2] Show-Jane Yen and Arbee L.P. Chen, "A Graph-Based Approach for Discovering Various Types of Association Rules," IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 13, NO. 5,pp. 839-845 SEPTEMBER/OCTOBER 2001.

[3]Ms.SanoberShaikh, Ms.MadhuriRao and Dr. S. S. Mantha "A New Association Rule Mining Based on Frequent Itemset," AIAA 2011,CS & IT 03, pp. 81–95 , 2011.

[4] K.L. Lee, Guanling Lee and Arbee L. P. Chen, "Efficient Graph-Based Algorithms for Discovering and Maintaining Association Rules in Large Databases," Knowledge and Information Systems (2001) 3: pp. 338–355, 2001.

[5] Hanbing Liu and Baisheng Wang, "An Association Rule Mining Algorithm Based on a Boolean Matrix," Data Science Journal, Volume 6, Supplement, 9 September 2007 pp. 559-565.

[6] Junfeng Ding, Stephen S.T. Yau, "TCOM, an Innovative Data Structure for Mining Association Rules Among Infrequent Items," Computersand Mathematics with Applications 57 (2009) pp. 290-301.

[7] R. Haralick, K. Shanmugam, I. Dinstein, "Textural Features for Image Classification," IEEE Transactions on Systems, Man, and Cybernetics (SMC-3) (1973) 610-621.

[8] G. Webb, "Efficient Search for Association Rules," International Conference on Knowledge Discovery and Data Mining, pp. 99-107, 2000.

[9] Fernando Berzal , Juan-Carlos Cubero, Nicolas Marin, "TBAR: An Efficient Method for Association Rule Mining in Relational Databases," Data & Knowledge Engineering 37, pp. 47-64 2001.

[10] De-chang Pi, Xiao-Lin Qin, Wang-FengGu, Ran Cheng, "STBAR: A More Efficient Algorithm for Association Rule Mining," Fourth International Conference on Machine Learning and Cybernetics, Guangzhou, 18-21 August 2005.