

SURVEY OF DIFFERENT DATA STRUCTURE BASED ASSOCIATION RULE MINING ALGORITHMS

Adhav Ravi¹, Bainwad A. M.²

¹M. Tech. Student, Dept. of CSE, SGGS IE&T, Nanded, India.

²Assistant Professor, Dept. of CSE, SGGS IE&T, Nanded, India.

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.

KEYWORDS: Association rule mining, Data structures, Graph, Matrix, Tree.

I. INTRODUCTION

Association rule mining [1], one of the most important and well researched techniques of data mining. 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 itemsets 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.

II. USE OF DIFFERENT DATA STRUCTURES IN ASSOCIATION RULE MINING

2.1. Graph

2.1.1. Primitive Association Pattern Generation (PAPG)

In this algorithm the first step is to construct association graph. This is two-step process numbering and graph construction. In the numbering phase, the algorithm PAPG [2] arbitrarily assigns each item a unique integer number. In the large itemset generation phase, PAPG scans the database and builds a Bit Vector (BV) for each item. The length of each bit vector is the number of transactions in the database. If an item appears in the i^{th} transaction, the i^{th} bit of the bit vector associated with this item is set to 1. Otherwise, the i^{th} bit of the bit vector is set to 0. The bit vector associated with item i is denoted as BV_i . The number of 1s in BV_i is equal to the support for the item i . For association graph construction PAPG uses Association Graph Construction (AGC) algorithm. The AGC algorithm is described as follows: For every two large items i and j ($i < j$), if the number of 1s in $BV_i \wedge BV_j$ 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 Large itemset Generation by Direct Extension (LGDE) 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 I_k 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 I_k to an item u , then the itemset (I_1, I_2, \dots, I_k) is extended into $k+1$ -itemset $(I_1, I_2, \dots, I_k, u)$. The itemset $(I_1, I_2, \dots, I_k, u)$ is a large $k+1$ -itemset if the number of 1s in $BV_1 \Delta BV_2 \Delta \dots \Delta BV_{i_k} \Delta BV_u$ achieves the minimum support. If no large $k+1$ -itemsets can be generated, the algorithm LGDE terminates.

2.1.2. Generalized Association Pattern Generation (GAPG)

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 POstorder Numbering (PON) 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. Here, we assume that all database items are specific items.

In the association graph construction phase, GAPG applies the algorithm Generalized Association Graph Construction (GAGC) 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 $BV_i \Delta BV_j$ 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

Undirected item set graph [3] is set of nodes $V(V_1, V_2, \dots, V_n)$ 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; $< V_1, V_2, \dots, V_n >$ 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 $T = \{T_{id}, \text{Itemset}\}$. 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 from V ;
 - 2.2. For each V_j ($j \neq i$)
 - If $(I_i \cap I_j) \neq \emptyset$ then
 - Add link between V_i and V_j
 - End if.
 - 2.3. Next.

3. Next

End

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 rules

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 of V_i ;
5. If count $((V_i, V_j) \geq \text{minsupp})$ then
 - 5.1. $L \cup V_j$;
 - 5.2. $L.\text{adjacentlist} = L.\text{adjacentlist} \cap (V_j.\text{adjacentlist})$;
 - 5.3. Call DFS (V_j) Procedure;
6. End if;
7. Confidence of each item is compared with minconf and strong association rules are generated.
8. End;

Procedure DFS (V_j):

Begin

1. If $V_j.\text{adjacentlist} \neq \emptyset$ then
 - 1.1. Select any other node, suppose V_k from $V_j.\text{adjacentlist}$;
 - 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 $L \cup V_k$.

Call DFS (V_k) ;
Output is frequent item set;
Delete V_k from V_j .adjacentlist;
Call DFS (V_j) ;

1.4. Else Return to its parent vertex V_j ;

1.5. Call DFS (V_j) ;

2. End;

Procedure *isloop* (L, V_k) :

Begin

1. If $V_k \in L$.adjacentlist then return V_k ;

2. Else delete V_k from V_j .adjacentlist;

3. CallDFS (V_j) ;

4. End;

5.

2.1.4. Direct Large Itemset Generation (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 as BV_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 result of $BV_{i1} \wedge BV_{i2} \wedge \dots \wedge BV_{ik}$, where the notation " \wedge " 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 $BV_i \wedge BV_j$ ($i < j$) 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 $\{I_1, I_2, \dots, I_k\}$ in L_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 item j , the itemset $\{I_1, I_2, \dots, I_{kj}\}$ is a candidate $(k + 1)$ -itemset. If the number of 1's in $BV_{i1} \wedge BV_{i2} \wedge \dots \wedge BV_{ik} \wedge BV_j$ 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 k^{th} iteration, the algorithm terminates.

2.2. Matrix

2.2.1. An Algorithm Based on Boolean Matrix (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 L_1 .

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 $T = \{T_1, T_2, \dots, T_m\}$ be the set of transactions and $I = \{I_1, I_2, \dots, I_n\}$ be the set of items. We set up a Boolean matrix $A_{m \times n}$, which has m rows and n columns. Scanning the transaction database D , if item I_j is in transaction 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 $A_{m \times n}$ 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 j^{th} column of the Boolean matrix $A_{m \times n}$. 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 $A_{m \times n}$ will be deleted from $A_{m \times n}$. 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' is 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 $A_{p \times q}$ has q columns where $2 < q \leq n$ and $h \leq p \leq m$, 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. Transactional Co-Occurrence Matrix (TCOM)

In order to employ the advantages of both horizontal and vertical layouts, uses matrix structure called TCOM [6]. 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 TCOM 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 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, $I = \{i_1, i_2, \dots, i_n\}$
2. Start with item in, for each item i_r in the set I, $1 \leq r \leq n$
 - 2.1. Let IS_{SET} be the set of itemsets, initially IS_{SET} is an empty set

- 2.2. Find out all existing item set $IS_A = \{i_{s_1}; i_{s_2}, \dots, i_r\}$ where

occurrence_count(i_{s_1})	\geq	occurrence_count(i_{s_2})	\geq	...	\geq	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 IS_A in the set IS_{SET}

- 3.1. For each item set IS_B in the set IS_{SET} where $IS_B \neq IS_A$

If IS_B contains IS_A

Let IS_C be the difference of IS_B and IS_A

If $occurrence_count(IS_B) \leq occurrence_count(IS_A) * \sigma$

$IS_A \rightarrow IS_C$ is a valid rule

Endif

End if

End for

- 3.2. Find out all happened item set IS_B where IS_B contains IS_A , and there exist at least one item j in IS_B with $occurrence_count(j) < occurrence_count(i_r)$

- 3.3. Find out the occurrence count for each IS_B found in step 3.2

- 3.4. For each itemset IS_B found in step 3.2

Let IS_C be the difference of IS_B and IS_A

If $occurrence_count (IS_B) \geq occurrence_count (IS_A) * \sigma$

$IS_A \rightarrow IS_C$ is a valid rule

End if

End for

End for

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 L_k consists of items $c[1], c[2], \dots, c[k]$, where $c[1] < c[2] < \dots < c[k]$. Associated with each item set are two fields: count field to store the support for this itemset, and index field to indicate the transactions that contain the itemset. The BitMatrix algorithm is described as:

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

In Step (1) of this algorithm, we initialize the bit-matrix 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, \dots, i_k in the j^{th} transaction, bits $a_{i_1j}, a_{i_2j}, \dots, a_{i_kj}$ 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 are 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 $p[1] = q[1] \wedge \dots \wedge p[k-2] = q[k-2] \wedge (p[k-1] < q[k-1])$ then {
3. $c = p \cup q$;
4. for all $(k-1)$ -subsets s of c do
5. if $(s \notin L_k)$ then {delete c ; $c = 0$; break;}
6. if $(c \neq 0)$ then {
7. $c.index = p.index \& q.index$;
8. compute $c.count$ from $c.index$;
9. if $(c.count > minsup)$ then $L_k = L_k \cup \{c\}$;
10. }//end if.
11. }//end if.

From Steps (1) to (5), the function simply helps generate the C_k that is a set of candidate k -itemsets. 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 (7).

2.2.4. Matrix Algorithm

Matrix algorithm[12] first generates matrix and then association rules are generated by using matrix. Procedure for matrix generation is as follows: It sets the items in I as columns and the transactions as rows of the matrix.

Let the set of items be $I = \{i_1, i_2, \dots, i_n\}$, and the set of the transactions be $D = \{t_1, t_2, \dots, t_m\}$. Then the generating matrix $G = \{g_{ij}\}$, $(i = 1, 2, \dots, m; j = 1, 2, \dots, n)$ is an $m * n$ matrix, where $g_{ij} = 0$ or 1 is determined by the following rule,

$$g_{ij} = \begin{cases} 1 & \text{if } i_j \in t_i, \\ 0 & \text{if } i_j \notin t_i. \end{cases}$$

After creation of matrix the candidate k -itemset is generated using the following process:

1. C_1 generation: At the beginning 1- itemset are generated and the support of the set $\{i_k\}$ is calculated by formula:

$$\text{support}(\{i_k\}) = \sum_{j=1}^m \langle g_j, S_k^1 \rangle,$$

where \langle, \rangle is the inner product of two row vectors and $g_j, j=1, 2, \dots, m$ are rows of the matrix G .

2. C_2 generation: Candidate 2-itemsets C_2 is the joint set of L_1 with itself, that is, $C_2 = L_1 \bowtie L_1$. The support of the set $\{i_k, i_j\}$ is :

$$\text{support}(\{i_k, i_j\}) = \sum_{s=1}^m \text{int} \left[\frac{\langle g_s, S_{i_k, i_j}^1 \rangle}{2} \right]$$

3. Repeat the above process with successively increasing number k until either C_k or L_k is empty. The support can compute according to the following formula:

$$\text{support}(\{i_{11}, i_{12}, \dots, i_{1k}\}) = \sum_{j=1}^m \text{int} \left[\frac{\langle g_j, S_{i_{11}, i_{12}, \dots, i_{1k}}^1 \rangle}{k} \right].$$

2.3. Tree

2.3.1. Tree Based Association Rule Mining(TBAR)

TBAR [9] is an Apriori based association rule mining algorithm 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:

1. set.Init (MinSupport);
2. itemsets = set.Relevants(1);
3. $k = 2$;
4. while $(k \leq \text{columns} \ \&\& \ \text{itemsets} \geq k)$
 - {
 - itemsets = set.Candidates (k);
 - If $(\text{itemsets} > 0)$
 - 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 C_k from L_{k-1} .

The itemset tree will look like

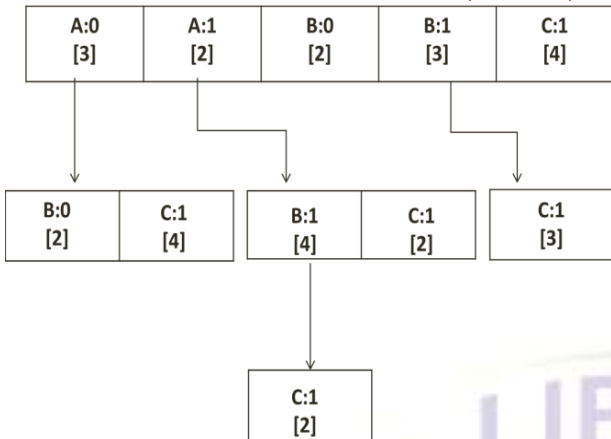


Fig.1 TBAR data structure example. After generation of itemset tree following algorithm will traverse the tree to find frequent itemsets and will generate association rules from those itemsets:

For each relevant k-itemset l_k in the tree with $k \geq 2$
 For each itemset $l_i \subset l_k$
 If $\text{support}(l_k) \geq \text{MinConfidence} * \text{support}(l_i)$
 Output rule $l_i \Rightarrow (l_k - l_i)$
 withconfidence = $\text{support}(l_k) / \text{support}(l_i)$
 andsupport = $\text{support}(l_k)$
 End If
 End For
 End For

2.3.2. Super TBAR (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 $\langle a, v \rangle$, but a 3-tuple $\langle a, v, t \rangle$, 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 item or not.

1. Generate all the frequent 1-itemsets, and store them in the 3-tuples.
2. According to the order $I_{n-1} \dots I_1$, generate itemset L_2 , validate whether it can constitute a frequent itemsets or not, and set the corresponding flag in the 3-tuple.
3. For each sub-itemset in L_2 , recursively generate L_n according to Step 2.
4. Seek for the tree's depth.
5. Find out the longest concatenations in the tree.
6. Produce all the association rules. This step is just as TBAR doing.

The STBAR datastructure will look like:

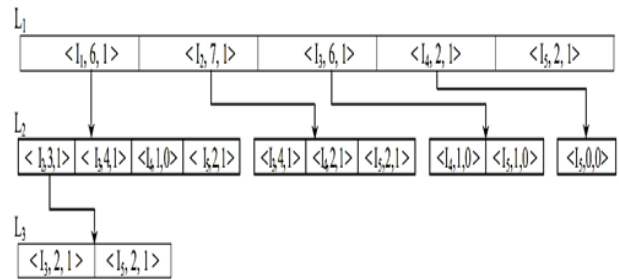


Fig.2 STBAR data structure example.

2.3.3. Trie Data Structure for Association Rule Mining

The data structure trie was originally introduced to store and efficiently retrieve words of a Dictionary. A trie [11] that stores the words mile, milk, tea, tee, teeny can be seen in Figure 3.

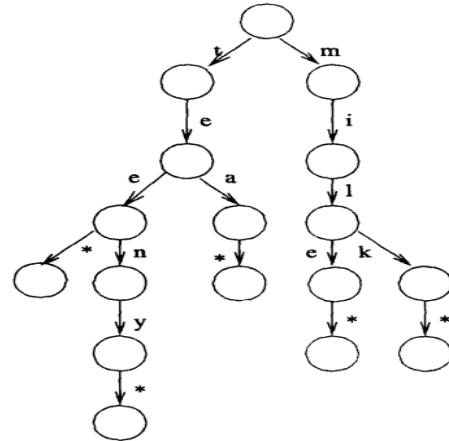


Fig.3 A trie containing five words.

Tries are suitable to store and retrieve not only words but any finite ordered sets. In this setting, a link is labelled by an element of the set, and the trie contains a set if there exists a path where the links are labelled by the elements of the set, in increasing order.

In our data mining context, the alphabet is the (ordered) set of all items I . A candidate k-itemset can be viewed as the word i_1, i_2, \dots, i_k composed of letters from I . Fig. 4 shows the trie storing frequent itemsets. Some of the itemsets are ACD, AEG, KMN, etc.

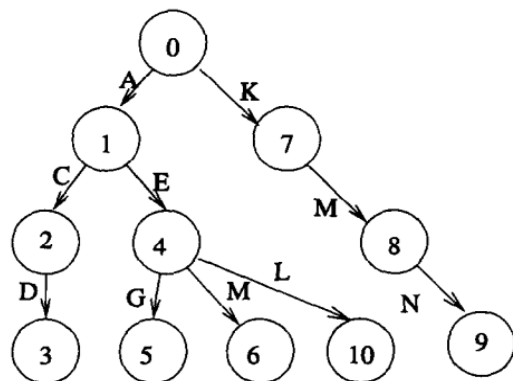


Fig.4 A trie containing five candidates.

III.CONCLUSIONS

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.

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, 1993, pp. 207-216.
- [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, 2001, pp. 839-845.
- [3] Ms.SanoberShaikh, Ms.MadhuriRao and Dr. S. S. Mantha, "A New Association Rule Mining Based on Frequent Itemset," AIAA 2011, CS & IT 03, 2011, pp. 81-95 .
- [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 3, 2001, pp. 338-355.
- [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," Computers and 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, pp. 610-621.
- [8] G. Webb, "Efficient Search for Association Rules," International Conference on Knowledge Discovery and Data Mining, 2000, pp. 99-107.
- [9] Fernando Berzal, Juan-Carlos Cubero, Nicolas Marin, "TBAR: An Efficient Method for Association Rule Mining in Relational Databases," Data & Knowledge Engineering 37, 2001, pp. 47-64.
- [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.

- [11] F. Bodon and L. Ronyai, "Trie: An Alternative Data Structure for Data Mining Algorithms," Mathematical and Computer Modelling 38, 2003, pp.739-751.
- [12] Yubo Yuan and Tingzhu Huang, "A Matrix Algorithm for Mining Association Rules," Advances in Intelligent Computing Volume 3644, 2005, pp 370-379.