A Dynamic Approach for Discovering Maximal Frequent itemsets

Geetha M
Department of Computer Science and Engineering
Manipal Institute of Technology
Manipal, Karnataka, India.
maiya_geetha@yahoo.com

R.J.D'Souza
Department of Mathematical &Computational Sciences.
National Institute of Technology Karnataka
Surathakal, Karnataka, India.
rjd@nitk.ac.in

Abstract— **We present a novel method, which reads the database at regular intervals as in Dynamic Itemsets Counting Technique and creates a tree called Dynamic Itemset Tree containing items which may be frequent, potentially frequent and infrequent. This algorithm requires less time to discover all maximal frequent itemsets since it involves a method for reducing the size of the database. This method prunes the transactions and items of the transactions which are not of our interest after every scan of the database. Also, this method is independent of the order of the items.**

Keywords- confidence; dynamic; frequent; prunning; Tree

I. INTRODUCTION

The Dynamic Itemset Counting Algorithm (DIC) [1] is an improvement over Apriori's candidate generation algorithm. The working of DIC algorithm is explained below as given in [2].

Initially certain "stops" are identified in the database. It is assumed that the records are read sequentially as in other algorithms, but pause to carry out certain computations at the "stop" points. Four different structures are used in this algorithm. They are (i) Dashed Box (ii) Dashed Circle (iii) Solid Box (iv) Solid Circle. Each of these structures maintains a list of itemsets. The itemsets in the "dashed" category of structures have a counter and stop number with them. The counter is to keep track of the support value of the corresponding itemset. The stop number is to keep track whether an itemset has completed one full scan over a database. The itemsets in the "solid" category structures are not subjected to any counting. The itemset in the solid box is the confirmed set of frequent sets. The itemsets in the solid circle are the confirmed set of infrequent sets. The algorithm counts the support values of the itemsets in the dashed structure as it moves along from one stop point to another. During the execution of the algorithm, at any stop point, the following events take place:

Certain itemsets in the dashed circle move into the dashed box. These are the itemsets whose support counts reach user defined value during this iteration.

Certain itemsets enter afresh into the system and get into the system and get into dashed circle.

The itemsets that have completed one full pass move from the dashed structures to solid structure.

Brin et al mentioned in their paper that this algorithm is implemented using Hash Tree used in Apriori algorithm[3], but involves some extra information stored at each node. It is a tree with the following properties. Each itemset is sorted by its items. Every itemset that is being counted has a node associated with it, as do all of its prefixes. The empty itemset is the root node. All the 1-itemsets are attached to the root node, and their branches are labeled by the item they represent. Every node stores (i) the last item in the itemset it represents (ii) a counter (iii) a marker as to where in the file counting is started (iv) its state and (v) branches if it is an interior node.

The following observations are made from the DIC algorithm.

In [2], it is mentioned that this algorithm needs four different data structures dashed box, Dashed circle, Solid Circle and Solid Box and each of these maintains a list of itemsets. If array data structure is used, then memory has to be allocated in advance to store candidate itemsets, frequent itemsets, and confirmed frequent and infrequent itemsets. In such cases, if the size of the database is very large then large number of candidate itemsets is generated and requires much space to store all of them. Since it is not known in advance how many of these are there, this approach does not seem to work well for large databases. If much memory is allocated in advance, then if there is less number of candidate itemsets, then most of the space allocated is wasted. Also, if user wants to know the frequent itemsets with respect to the reduced minimum support then infrequent set has to be scanned. This takes much time.

If it is implemented as it is given in [1] following things have to be observed. Given a collection of itemsets, the form of hash tree structure is heavily dependent on the sort order of the items. During the first interval of M transactions only 1-itemsets are counted and the tree structure does not depend on the order. After the first interval, the order of the items is changed and tree is built from there. This technique incurs some overhead due to the re-sorting.

This motivated us to propose the following simpler method. This method reads the database at regular intervals as in DIC and creates a tree containing items which may be frequent, potentially frequent and infrequent. This algorithm requires less time to discover all maximal frequent itemsets since it involves a method for reducing the size of the database. This method prunes the transactions and items of

978-0-7695-3521-0/09 $25.00 © 2009 IEEE
DOI 10.1109/ICCET.2009.153

the transactions which are not of our interest after every scan of the database. Also this method is independent of the order of the items.

II. PROPOSED METHOD

Structure And Generation of Dynamic Itemset Tree

The proposed method uses the concept of a tree called Dynamic itemsets tree DT and allocates memory at run time. Every node consists of four parts

Item number
count
stop number
status of the node i.e whether particular item is confirmed frequent(Cf) / frequent(f) /infrequent(if) /active(ac) and two pointers, child and sible pointers.

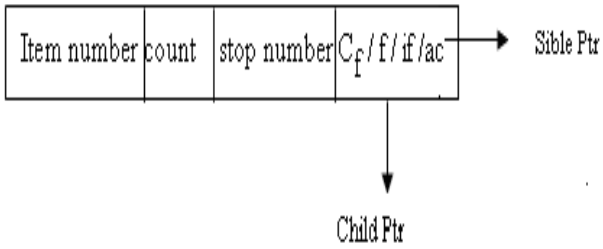

Figure1 . Node Structure

Let D be a given transaction database containing n items and assume m to be the number of stop points. Initially, the tree DT contains n nodes; each corresponds to every 1– item and status of the node is assumed to be 'ac' i.e. active. The transactions are read sequentially one after another. For every item in the read transaction, its counter is incremented in the tree DT. This process is continued until the next stop point is reached. Before the next set of transactions are read, certain computations like finding itemsets which satisfy user defined minimum support are identified and also possible candidate itemsets are generated. A new node is created for every element of un matching part of candidate itemsets in D_T . This node is then added to the tree DT T as a child of last matching item of the candidate itemset with status as active, the count value as 0 and stop number as the current stop number. If some of the itemsets become frequent before it has completed one pass of the database then its status has to be changed to 'f' i.e. frequent.

The itemsets if they have completed one full pass then their status is made ' Cf' or ' if ' depending on whether they are confirmed frequent or infrequent respectively. This procedure is continued until the status of all nodes in D_T become either'C_f ' or 'if '.

Pruning Technique

Once the complete scan is made for some itemsets, they will be either confirmed to be frequent or infrequent itemsets. From this step onwards all the transactions which contain infrequent itemsets are pruned from the database in order to avoid scanning the irrelevant items in the database. The following points are to be noted in this case.

(i) The counting for all 1 - itemsets starts from the beginning of the database. These itemsets complete one full pass at the end of the database. Therefore, during the second scan of the database, while constructing the Dynamic Itemset Tree by reading the transactions interval wise, items which are not frequent in every transaction read are removed.

(ii) For all other itemsets once they complete the full pass, in the next iteration, the irrelevant elements are removed from the database.

If {A, B, C} is a candidate large 3- itemset then all its subsets have to be frequent. Therefore, all its 2-item subsets should belong to L2, the set of all frequent 2-itemsets. This fact suggests that a transaction can be used to determine the set of large (k+1)-itemsets only if it consists of (k+1) large k-itemsets in Lk obtained in the previous pass. This means that, if the number of candidate itemsets is very close to that of large itemsets, while counting k-subsets, the transactions can be efficiently trimmed by eliminating items which are found to be irrelevant for later large itemset generation.

Also, if a transaction contains some large (k+1) itemsets, any item contained in these (k+1) itemsets will appear in at least k of the candidate k-itemsets in Ck. As a result, an item in transaction t can be trimmed if it does not appear in at least k of the candidate k–itemsets in t. For example,

(i) If L1= {2, 3, 5, 6, 7} and t = 2, 3, 4
Only large subsets for this transaction are {2}, {3}. Therefore, this transaction can be used for determining candidate 2-itemsets since it contains 2 large 1-itemsets from the previous iteration.
Also, count [2]=1, count[3] =1 and count[4] =0. Since the item 4 does not satisfy the necessary condition, it has to be deleted from this transaction.

(ii) If L2 = { { 2, 3}, {2, 6}, {5, 7}} and t =2,3,5,6,7
Only large subsets for this transaction are {2, 3}, {2, 6}, {5,7}
Therefore, this transaction can be used for determining candidate 3-itemsets since it contains 3 large 2-itemsets from the previous iteration.
Also, count [2]= 2, count[3]= 1, count[5]= 1, count[6]= 1 and count[7]= 1.
As items 3, 5, 6, 7 do not satisfy the necessary condition, they are to be deleted from the transaction. Therefore, transaction size is reduced.

(iii) If L2 ={{ 2, 3}, {2, 6}, {5, 7}} and t =2,3,5
Only large subsets for this transaction is {2,3}

Since this transaction does not have sufficient number of 2-large itemsets to find 3-large itemsets, this entire transaction from the database can be deleted.

Dynamic Itemsets Tree Algorithm

(i) Construction Dynamic Itemsets Tree

Input:

The given database D, the tree D_T containing nodes corresponding to every 1- itemsets. Current stop-number = 0; status ='ac', s = user defined minimum support

Output:

The tree D_T with frequent and infrequent itemsets.

Do until status of all the nodes is equal to 'if ' or 'C_f'

begin

for every transaction t in D till the next stop do

begin

if any infrequent itemsets found

prune t in D

remove nodes from D_T , corresponding to the irrelevant items.

increment the counters of items of t in D_T

end

increment the current-stop-number by 1

for each itemset I in D_T do

begin

if status = 'C_f' , flag=0; continue.

else

if count [I] $\geq$ s and status ='ac'

change the status field of this itemset to 'f '

generate new itemsets and put them into tree by extending the prefixes of those itemsets with counter value=0 ;

stop_number=current stop-number and status ='ac'

else if status='ac' and stop_number=current stop _number

set status ='if '

if status ='f ' and stop_number=current _stop_number

set status = 'C_f'

end

if (flag= 0) then return tree D_T

else

continue;

end

(ii) Reducing Dynamic itemset Tree to contain only maximal frequent itemsets

Input: The Dynamic itemset Tree D_T

Output: The Reduced Dynamic itemset Tree D_T with only maximal frequent itemsets.

Traverse D_T in preorder

for every frequent itemset I in D_T do

begin

if I is contained in any superset g in D_T

delete node corresponding to I

end

(iii) Discovery of maximal frequent itemsets

Input: The Dynamic Itemset Tree D_T with only maximal frequent itemsets.

Output: Maximal frequent itemsets MF

Initially, MF = { φ }

Traverse D_T from root to leaf in preorder

begin

for every maximal path Mp from root to leaf do

begin

MF = MF $\cup$ Mp

end

end

III. THEORETICAL ANALYSIS

A Construction Dynamic Itemsets Tree

(i) The construction of Dynamic Itemset Tree requires a minimum of one database scan, same as that of Dynamic Itemset Counting Technique.

(ii) The number of transactions scanned by this method is clearly less than DIC algorithm since it employs a pruning technique to reduce the size of the transaction database at every pass of the database. If there are N transactions and if DIC algorithm requires k scans of the database to discover all frequent itemsets then Dynamic Itemset Tree algorithm requires less than or equal to k scans but takes less time to achieve the same.

(iii) The candidate generation step requires the same amount of time as that of DIC algorithm. But due to pruning technique the candidate itemsets which are not of interest are deleted from the tree. i.e for every pass if m itemsets are found to be infrequent, then at least m nodes are to deleted from the tree. If the algorithm requires k scans to discover all maximal frequent itemsets then at least km nodes are deleted from the tree. Therefore removal of irrelevant items is in ***O***(km).

(iv) If infrequent items are not deleted from the tree then the Dynamic Itemsets Tree D_T constructed will contain both frequent and infrequent itemsets with count. Therefore, it is possible to discover all maximal frequent itemsets, border set and closed itemsets at different user defined minimum support. This support can even be Reduced Minimum Support. But in this case, since irrelevant items remain in the tree, these m irrelevant nodes are scanned at every stop. If there are k

database scans and j stops for one scan, then the tree has to be scanned (jmk) times. Therefore this step is in ***O***(jmk). Hence this is greater than the time required for deletion of irrelevant nodes.

(v) For a given database, if DIC creates n nodes in k passes, then this algorithm creates (n- km) nodes of same size in k passes. Hence the space utilized by Dynamic Itemset Tree is less when compared to DIC algorithm and requires less time to scan D_T than hash tree constructed by DIC.

B Reducing Dynamic itemset Tree to contain only maximal frequent itemsets

This step depends on the number of leaf nodes. If there are m leaves in the tree, then the number of comparison operations performed to reduce the tree to contain only maximal frequent itemsets are $\left(\frac{m(m+1)}{2}\right)$ in addition to m tree scan operations are required. Therefore this step is in ***O***(m^2)

C Discovery of maximal frequent itemsets

This step depends on the number of leaf nodes in the tree D_T . If there are m leaf nodes in D_T then there are m maximal frequent paths in a tree. Further this step requires only one scan of D_T. If there are n nodes in a tree then number nodes scanned in this step is in ***O***(n).

IV. ILLUSTRATION

Consider a sample database shown in Table 1 with the user defined minimum support =2 transactions, number of stops=2 and number of transactions per stop =2.

Table 1 . Sample Database for the construction of Dynamic Itemsets Tree

Transaction ID	Item Numbers
T1	1, 2, 3, 4
T2	1, 2, 3, 5
T3	1, 4
T4	1, 3, 4

Applying the method to the above sample database, without removing infrequent candidates, the Dynamic Itemsets Tree obtained is shown in the Figure 1. The tree in Figure 2 with 12 nodes contains all the information required to find maximal frequent itemsets, infrequent itemsets for different user defined minimum support. For this purpose, initially sufficiently small user defined minimum support s can be chosen, so that it will possible to find maximal frequent itemsets for all values of thresholds greater than s. In the Figures 2 and 3, n* indicates that an itemsets has entered the tree at the n^{th} stop in the second scan of the database.

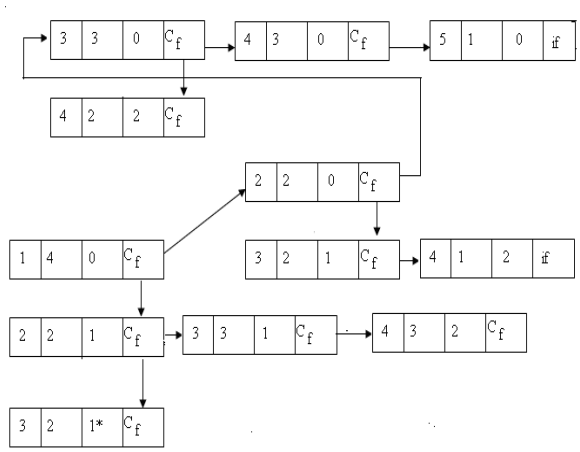

Figure 1 . Dynamic Itemsets Tree with frequent and infrequent itemsets corresponding to Table 1

From the above tree it can be seen that the itemsets {5}, {2, 4} are infrequent itemsets. Hence deleting the nodes corresponding to them reduces the size of the tree. Since Dynamic Itemsets Tree method attempts to discover only maximal frequent itemsets, only nodes corresponding to the maximal frequent itemsets are to be kept on the tree. Hence the nodes corresponding to the itemsets {4}, {1, 3} and {2, 3} are deleted from the tree. The final tree with respect to sample database given in the Table 1 is shown in Figure 3.

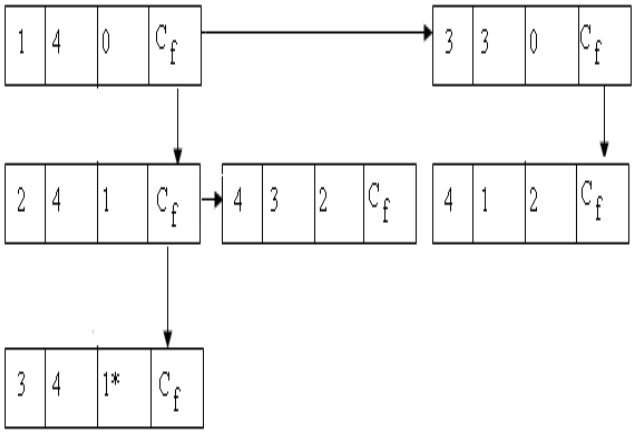

Figure 2. Dynamic Itemsets Tree for the Table 1

Since Dynamic Itemsets Tree algorithm involves the technique for reduction of tree and database size , it can also be applied for both sparse and large databases. The algorithm DIC is not recommended for large data sets since it involves storing of large number of candidate itemsets , which results in space overhead problem.

V. CONCLUSION

This new method for discovering maximal frequent itemsets from large as well as sparse databases does not depend on order of the items. It has been proved theoretically that this method is space, time efficient and works for increased minimum support.

This approach right now does not work for reduced minimum support. i.e if minimum support is reduced, then the tree does not contain enough information to discover all maximal frequent itemsets with respect to that reduced minimum support. This can be achieved using the Dynamic Itemset Tree with infrequent and frequent items. But this involves space overhead problem. Therefore, as a future enhancement, a variation of the method could be developed to handle Reduce Minimum Support efficiently.

VI. REFERENCES

[1] Sergey Brin, Rajiv Motwani , Jeffrey D.Ullman , Shalom Tsur(1997) " Dynamic Itemset counting and implication Rules for Market Basket Data", ," Proc. ACM SIGMOD Conf. Management of Data..Canada,255-264.

[2] Arun K Pujari(2003)– "Data mining Techniques": Universities Press(Indis) Private Limited. Hyderbad, India.

[3] Rakesh Agarwal, Ramakrishnan Srikant(1994), "Fast algorithms for mining association Rules", In proceedings of the 20th International Conference on Very Large databases, Santigo, 478-499

[4] J. Han and M. Kamber(2004), "Data MiningConcepts and Techniuqes": San Franscisco, CA:. Morgan Kaufmann Publishers