Discovery of frequent itemsets using weighted tree approach

PREETHAM KUMAR[1], ANANTHANARAYANA V.S[2]

[1]Department of Information & Communication Technology, M.I.T., Manipal, Karnataka, India

[2]Professor and Head, Department of Information Technology, N.I.T.K., Surathkal, Karnataka, India.

Summary

Most of the association rules mining algorithms to discover frequent itemsets do not consider the components of transactions like its quantity or weight or its total profit. In a large database it is possible that even if the itemset appears in a very few transactions, it may be purchased in a large quantity for every transaction in which it is present and may lead to very high profit. Therefore the weight is the most important component and without which it may lead to lose of information. Our novel method discovers all frequent itemsets based on its weights in a single scan of the database. In order to achieve this, we first construct a weighted tree containing weights and a set of transaction id's corresponding to every attribute in the database. Then by scanning the above tree we can discover all frequent itemset. This method is also found to be efficient than FP-tree, which require two scans of the database to discover all frequent itemsets based on user defined minimum support. Further, using Weighted tree constructed for the above method is used for discovering best order or sequence of attributes. If database is read in this best sequence , we can construct an abstraction of the entire database in the memory which occupies less space in the memory when compared to sequential reading.

Index Terms—Association rule, Data mining, frequent itemsets, Support, Weighted Support, Weighted tree.

I. INTRODUCTION

The goal of knowledge discovery is to utilize those existing data to find out new facts and to uncover new relationships that were previously unknown, in an efficient manner with minimum utilization of the space and time. At first, our method discovers all frequent itemsets based on weighted minimum support by constructing the tree called Weighted tree in an efficient manner. Next our method uses Weighted tree and then discovers the best sequence or order of the attributes, in which database is read, an abstraction of the database can be constructed with a minimum utilization of the space.

A. Weighted frequent itemsets

Mining association rules is an important branch of data mining, which describes potential relations among data items (attribute, variant) in databases, the well-known Apriori algorithm [1] was proposed by R. Agrawal et al., in 1993. Mining association rules can be stated as follows: Let I={i1, i2, … im } be a set of items. Let D, the task-relevant data, be a set of transactions, where each transaction T is a set of items such that $T \subseteq I$. The quantities of items bought are not considered. Each transaction is assigned an identifier, called TID. Let A be a set of items, a transaction T is said to contain A if and only if $A \subseteq T$. An association rule is an implication of the form $A \rightarrow B$, where $A \subseteq I$, $B \subseteq I$, and $A \cap B = \emptyset$. The rule $A \rightarrow B$ holds in the transaction set D with support s, where s is the percentage of transactions in D that contain AUB (i.e., both A and B). This is taken to be the probability, $P(A \cap B)$. The rule $A \rightarrow B$ has confidence c in the transaction set D if c is the percentage of transactions in D containing A that also contain B. This is taken to be the conditional probability, P(B|A). That is, $\text{Support}(A \rightarrow B) = P(A \cap B) = s$, $\text{Confidence}(A \rightarrow B) = P(B|A) = \text{Support}(A \rightarrow B)/\text{Support}(A) = c$.

Mining of association rules is to find all association rules that have support and confidence greater than or equal to the user-specified minimum support and minimum confidence respectively [2]. This problem can be decomposed into the following sub problems:

Manuscript received August 5, 2008.
Manuscript revised August 20, 2008.

a) All itemsets that have support above the user specified minimum support are discovered. These itemset are called the frequent itemsets.
b) For each frequent itemset, all the rules that have user defined minimum confidence are obtained. The second sub problem, i.e., Discovering rules for all given frequent itemsets and their supports, is relatively straightforward as described in [1].
There are many interesting algorithms for finding frequent itemsets based on user defined minimum support. One of the key features of all algorithms is that each of these methods assumes that the underlying database size is enormous, and involves either candidate generation process or non-candidate generation process. The algorithms with candidate generation process require multiple passes over the database and are not storage efficient. In addition, the existing algorithms discover all frequent itemsets based on user defined minimum support without considering the weights such quantity, cost and other attributes which lead to profit.
If an itemset satisfies user defined weighted support then we say that it is weighted frequent itemset.
Consider for example a sample database given in Table 1 in which every element of each transaction represents either quantity or cost of the respective attribute or item.

Table 1: Sample Database

TID\Atrributes	A	B	C	D
1	10	5	0	0
2	0	0	3	0
3	4	0	4	0
4	5	2	5	0
5	0	0	0	10

It may so happen that an itemset appears in a very few transactions in a large quantity or cost which leads to profit will not qualify as frequent itemset based on user defined minimum support. This results in a lose of information. In the sample database given in the Table 1,if user defined minimum support is 2 transactions then an item D is not frequent and will not appear in the set of frequent itemsets even though if it is bought in large quantity and leads to more profit than other frequent items. This motivated us to propose the following method which discovers all frequent itemsets by based on user defined minimum weight such as quantity, cost etc.

B. Optimal order of the attributes

If a database is very large then reading and representing it in the memory requires much time and space. In order to achieve this, we find the best order of attributes in which it is read, an abstraction of entire database like PC –tree [4]can be constructed with a minimum utilization of memory.
For example, consider the tree construction for the sample database in Table 2

Table 2: Sample Database

	A1	A2	A3
T1	3	2	1
T2	6	2	1
T3	2	2	4

The dataset given in Table 2 has three attributes. Hence, a total of 3! i.e. 6 different combinations of attribute-orders are possible. The tree structures, for 3 of the 6 possible combinations of attribute-order, are shown in Figure 1. Note that the elements of a particular attribute are found at the level specified by the position of that attribute in attribute-sequence. Also, take a note of the total number of nodes, i.e. the value of **n**, in the trees obtained for different combinations of attributes. Thus, our aim is to determine the order of attributes that leads to a tree with the minimal number of nodes in the tree. In this case (for dataset in Table 2), the best optimal order is 2-3-1.

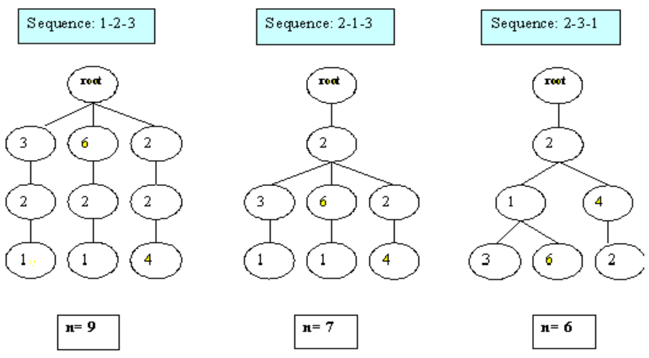

Figure 1: Different Forms of Tree

Finding the best sequence of attributes for a dataset by checking for all the possible combinations is an exhaustive and time-consuming process. As the number of different attributes in the database becomes very large, this approach of checking for all combinations becomes prohibitive. Hence, we came up with an approach that obviates the need to iteratively construct the tree for all combinations of attributes in order to find an optimal sequence.

II. PROPOSED METHOD

Structure of Weighted tree

The Weighted tree has two different nodes and is shown in the Figure 2.

(i) The first type of node labeled with attribute contains attribute name and two pointers, one pointing to the nodes containing transaction ids and weights and another is a child pointer pointing to the next attribute. This node represents the head of that particular branch.

(ii) The second type of node has 2 parts. First part is labeled TID represents a transaction number or id and second part of which is labeled weight, indicates quantity purchased in that transaction or cost or other components. This node has only one pointer pointing to the next object having this particular attribute.

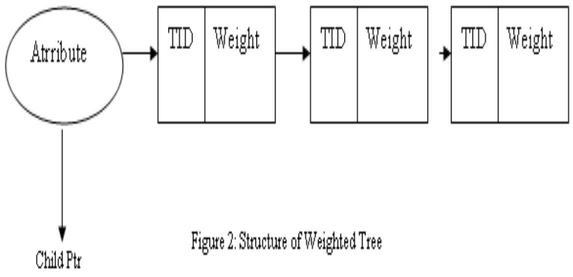

Figure 2: Structure of Weighted Tree

A Weighted Tree for the Sample Database given in the Table 1 is shown in Figure 3.

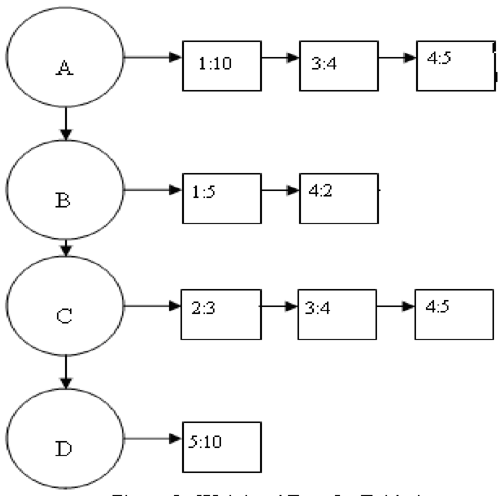

Figure 3 : Weighted Tree for Table 1

The Weighted tree algorithm involves 3 steps. They are
A. Construction of Weighted Tree
B Removal of infrequent attributes of Weighted Tree
C. Discovery of frequent itemsets based on weights

A. Algorithm for constructing Weighted Tree

```
Input: The database D
Output: Weighted tree
For each attribute weight w in a transaction t € D do begin
  create a node labeled with w and add these nodes to the respective attribute node.
end
```

B. Algorithm for Reducing the Weighted Tree

Input : **w_min_sup** =user specified weighted minimum support
Output: Weighted tree without infrequent attributes.

```
for each attribute in a Weighted tree do
    begin
      If sum(weights of all nodes) < w_min_sup then
            remove that branch from the tree
    end
```

For example, In the above case if we consider min_sup=3 then only attributes A and C are frequent in the database. The attributes B and D are found to be infrequent.
If we consider **weighted_minimum_support** = 10 then the attributes A, C and D will be frequent in the database.

C. Algorithm to discover all frequent itemsets

```
Input: A reduced Weighted tree, **w_min_sup**=user specified weighted minimum support
Output: set of all frequent itemsets. $F_w$
F= {set of all frequent attributes}
P=set of all non empty subsets of F excluding the sets containing one attribute.
Begin
$F_w$ = { set of all frequent attributes or one itemset }
for each f in P do
 begin
        T={TIDs of first attribute in f }
for each m in f other than first attribute do
begin
      T=T ∩ { TIDs m}
End
  If T is non empty then
     If Sum of weights of T >= w_min_sup
        $F_w = F_w \cup f$
end
```

Illustration:

Consider for example, a sample database given in Table 1, If **w_min_sup =10 then applying above algorithm, we get F_w** ={A, C, D}
P={ {A, C}, {A,D}{C,D}, {A, C, D} }
Consider a set {A, C} , which appears in transactions 3 and 4. i.e T={3,4} . Also, sum of weights=9+9=18. Hence it is frequent.
By similar arguments, we found that the sets {A, D}, {C,D} and {A,C, D} are infrequent sets.
Hence **F_w** = { {A}, {C}, {D},{ A, C}}.

D. Discovery of best order of attributes

To compute the best order of the attributes, we first fill the all the entries of matrix defined below by scanning the Weighted tree. Then, we will get the optimal tree if we start with an attribute that has least number of nodes under it. The order of remaining attributes can be determined by taking into account only the number of links between two attributes. For easy understanding, we can visualize this approach as analogous to Prim's algorithm for finding the minimum spanning tree.

Defining matrix

If the number of attributes in the database is N, then take a matrix of size N*N i.e. Mat[N][N].

```
For  0< i,j <=N,    where i and j represent the attribute numbers.
If (i= =j)
  Mat[i][j] = number of nodes under the attribute
        number i.
     // Mat[i][i] stores the number of nodes
        under attribute Ai.
Else
   Mat[i][j]= number of outgoing links from
header number i to header number
```

Determining the sequence

Data structure used here are
Order[N]= Array of size N, storing the order of attributes to be considered for obtaining the optimal tree.

Node_min = Mat[i][i], where i is the attribute number of attribute having the least number of nodes under it.

Row_min(k) = min (Mat[i][j]), where i=k , and j is such that it is not equal to k and not already entered in Order[N].

Attr_order[N] : contains the final order of the attributes to be considered
node_span=denotes the possible number of nodes at a given level in the tree.

attr_count : the number of attributes already considered for the sequence in Order[].

t_count : value denoting likely number of nodes to occur in a tree constructed for a sequence.

Final_tcount : the t_count value associated with sequence in Attr_order[];

```
For each attribute Ai such that Mat[i][i]= node_min
 { attr_count=1;
  Order[attr_count]=i;
Mat[i][i]=t_count=node_min;
  Call min_links (i, Order[], attr_count, Mat[i][i],
   Mat[i][i]); }
```

```
**Procedure min_links** ( attribte number i, Order[N],attr_
count, node_span, t_count )
{
 If attr_count equal to N then
  {
    If t_count is less than Final_tcount or if
    Attr_order[] is empty then
     {
       Update Final_tcount to t_count;
       Update Attr_order[] with entries in Order[];
     }
   Return;
  }

For each j in the ith row of matrix Mat
If j is not equal to i as well as j is not already
contained in Order[]
  {r_min = Row_min(i) ;

   If  r_min >nodespan
        nodespan=r_min;
          t_count= t_count + node_span;
        attr_count++;
       Order[attr_count]=j;

      Call min_links (j, Order[], attr_count,
        node_span, t_count);
    }}
```

Illustration:
Let us consider the sample database given in Table 2 which has 3 attributes and 3 tuples.
The weighted tree corresponding to the Table 2 is given in the Figure 4.
While discovering the number of nodes under any attribute, if we have same weights for 2 different nodes under any attribute, we just create one in order to save space in memory. Hence, it follows from the Figure 4 that there exist 3 nodes under A1 with different weights and 1 for A2 and 2 for A3. Association among the attributes is represented in the matrix defined by Table 3.

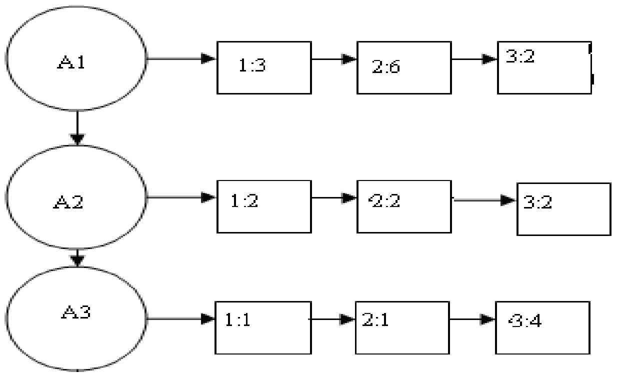

Figure 4 : Weighted Tree for Table 2

	A1	A2	A3
A1	3	3	3
A2	3	1	2
A3	3	2	2

Table 3: Matrix for figure 4

Finding the sequence: (step 3 of algorithm)

It can be observed that the row number as well as the column number in the matrix denote the Header number (attribute number).

After scanning the matrix diagonally, we get Node_min = 1 and corresponding value of i as 2. thus,
i=2;
t_count = 1;
node_span=1.
attr_count=1.
So, Order[1] = 2.
One thing to be noted is that, once we have considered an attribute number i.e.header number in the array Order[], we cannot consider column number corresponding to that header numbers, for finding the Row_min value in the specified row, besides the exclusion of the position in the row where row number and column number are same as that position stores the node number, not the number of links.

Since i=2, we go to 2rd row of the matrix and call the function Row_min(2) to get minimum available row_min value in the 2rd row i.e. row_min = 2, corresponding to i=3. Hence,
i=3
Order[2] = 3.
attr_count =1+1=2;
Since row_min > node_span.
Node_span = 2
New t_count = 1+2 = 3. (i.e. previous t_count + node_span).

Now we go to row number 3 and find the available row_min value. Here, we note that the minimum available row_min value of 3 occurs at all positions. We take i=1 since i=2 and 3 already considered.
Order[3]=1
attr_count=2+1=3
Since row_min > node_span.
Node_span = 3
Row_min=3.

t_count = 3+3 = 6
At this point attr_count equals N, and thus a complete sequence is obtained in the array Order[] as shown below.
$2 \rightarrow 3 \rightarrow 1$
Thus the final sequence obtained in Attr_order is:

Final Sequence: $2 \rightarrow 3 \rightarrow 1$.
i.e rding the database in the above order produces minimum number of nodes.

III. PERFORMANCE ANALYSIS

A. Weighted frequent itemsets

For the performance analysis, simulation of buying patterns of the customers in retail patterns is generated and in the

data set which we used every element of the transaction is considered as quantity of the corresponding attribute in the database. We have compared our algorithm with FP- tree and we found that ours is more time efficient.
The above algorithm is implemented and used for discovering frequent itemsets for data sets containing the transactions 100, 500, 1000, 5000 with 20 attributes and time required to discover all frequent itemsets is shown the figure 5.

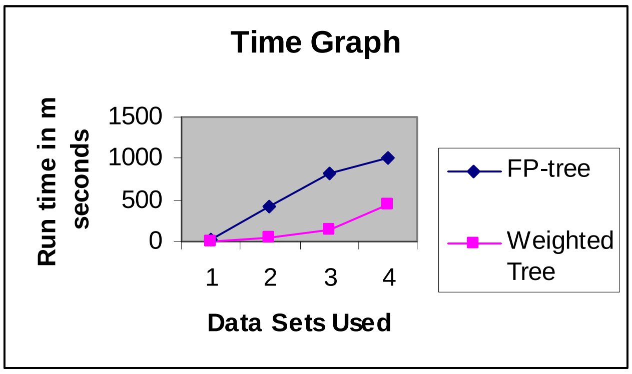

Figure 5: Time Graph

The proposed algorithm excels the FP-tree algorithm in three ways:

1. Scans database only once
2. No sorting of each item of the transaction.
3. No repeatedly searching the header table for maintaining links, while inserting a new node into tree.
4. Considers most important components like quantity, cost etc.

B. Best Sequence of attributes

In order to test the algorithm for discovering best order of attributes against a dataset having a large number of attributes i.e. having a large number of possible combinations of attributes, we used SPECTF Heart Data[5], that contained 45 attributes and had 187 tuples i.e N= 45 and Tn=187 .

The best sequence of attributes obtained is as follows: 0(36)—31(126)—32(133)—34(138)—21(141)—12(145)—17(136)—18(144)—7(152)—9(148)—10(151)—4(143)—3(150)—8(151)—14(147)--13(151)—22(153)—5(152)—33(155)—27(154)—2(164)—30(148)—29(155)—1(161)—37(152)—38(159)—15(156)—20(159)—6(159)—16(160)—36(160)—25(175)—40(171)—41(161)—43(164)—44(165)—42(166)—26(167)—24(159)—23(168)—39(171)….

The values in bracket along with the attribute numbers in the sequence denote the total number of nodes obtained.

The results of the experiment are shown in the following Table.

Table 4. Experimental result obtained with SPECTF Heart Data

Avg. time taken to construct a tree for any sequence (T0) in sec.	1.2 sec
Total time taken to find the best sequence for optimal tree (T1 in sec.) using iterative checking method	(45!)* 1.2 = 1.722*10^56 sec
Time taken to find the best sequence for optimal tree (T2 in sec.) using our algorithm	4.3 sec
Total number of nodes in the tree using serial order of attributes (S1)	7729 nodes
Total number of nodes formed in the tree using the best sequence of attributes obtained using our algorithm	6651 nodes
Saving in nodes(space) S= S1-S2	1078 nodes

We observe that the sequence leading to optimal tree leads to a saving of 1078 nodes (space), as compared to a tree constructed using the serial order of attributes (no sorting of elements required in this case). But, the best part is the huge saving in time that is achieved by this algorithm to find the sequence for obtaining optimal tree structure.

IV CONCLUSION

The Weighted Tree Algorithm is a new method for finding frequent itemset based on user defiend weights and is found to be efficient when compared to FP-tree. Also we found a method for representing the database in a memory which consumes less space. As such, we are still working on it with the aim of extending the application of this algorithm to various kinds of databases.

REFERENCES

[1] J. Han and M. Kamber. Data Mining, Concepts and techniques. Morgan Kaufmann publishers, 2001
[2] J. Han, J. Pei, and Y. Yin. *Mining frequent patterns without candidate generation.* In *ACM-SIGMOD*, Dallas, 2000.
[3] R. Hemlata, A. Krishnan, C. Scenthamarai, R. Hemamalini. *Frequent Pattern Discovery based on Co-occurrence Frequent Tree.* In Proceeding ICISIP-2005.
[4] Ananthanarayana V. S, Subramanian, D.K., Narasimha Murthy, M- *Scalable, Distributed and Dynamic Mining of Association Rules using PC-Tree*; p559- 66, HIPC, Bangalore, INDIA, 2000.
[5] SPECTF Heart Data
[6] IBM/Quest/Synthetic data.