International Journal of Electronics Communication and Computer Engineering
Volume 3, Issue 1, ISSN (Online): 2249–071X, ISSN (Print): 2278–4209

Computational Model for Mining Frequent Sets in Large Databases

Anju Singh
BUIT, BU, Bhopal
asingh0123@rediffmail.com

R. C. Jain
SATI, Vidisha
dr.jain.rcd@gmail.com

Abstract **- In this paper we have addressed the problem that we overcome in data mining applications i.e. mining frequent patterns in large databases efficiently in less time with less memory requirement. We are proposing a computational model for finding frequent patterns in large datasets with less number of scans .We have also proposed methodology which would help in storing large database compactly and thus help in improving the storage space requirement. Our model would help in generating less patterns and thus improving the mining time required in large databases. Our computational model is an amalgamation of three approaches i.e. bottom up counting inference and top down intersection method for generating the frequent sets and tree based approach for storing the databases compactly.**

Keywords **- Computational Model, Mining, Frequent Sets, Large Databases, Survey.**

I. INTRODUCTION

Mining frequent patterns from large transactional databases plays an essential role in many data mining tasks and have broad applications. Knowledge discovery in databases (KDD) has received increasing attention and has been recognized as a promising new field of database research. It is defined by Fayyad et al. [1] as "the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data". The key step in the knowledge discovery process is the data mining step, which is "consisting of applying data analysis and discovery algorithms that, under acceptable computational efficiency limitations, produce a particular enumeration of patterns over the data" [1].

Discovering association rules that identify relationships among sets of items in a transactional database is an important problem in Data Mining. Finding frequent item sets has been an active research area since it is the crucial step in association rule discovery. However, efficiently mining frequent item sets from dense datasets is still a challenging problem.

A key component of many data mining problems is formulated as follows. Given a large transactional database of sets of items (representing market basket data, alarm signals, etc.), discover all frequent item sets (sets of items), where a frequent item set is one that occurs in at least a userdefined percentage (minimum support) of the transactional database. Depending on the semantics attached to the input database, the frequent item sets, and the term "occurs".

II. LITERATURE SURVEY

Three approaches have been studied for mining frequent patterns: The first is traversing iteratively the set of all patterns in a level wise manner [2]. During each iteration corresponding to a level, a set of candidate patterns is created by joining the frequent patterns discovered during the previous iteration, the supports of all candidate patterns are counted and infrequent ones are discarded. The most prominent algorithm based on this approach is the Apriori algorithm [3] that uses identical properties as the OCD algorithm [4] proposed concurrently.

The second approach is based on the extraction of maximal frequent patterns, from which all supersets are infrequent and all subsets are frequent. This approach combines a level wise bottom-up traversal with a top-down traversal in order to quickly find the maximal frequent patterns. Then, all frequent patterns are derived from these ones and one last database scan is carried on to count their support. The most prominent algorithm using this approach is Max-Miner [5]. Experimental results have shown that this approach is particularly efficient for extracting maximal frequent patterns, but when applied for extracting all frequent patterns performances drastically decrease because of the cost of the last scan which requires roughly an inclusion test between each frequent pattern and each object of the database.

The third approach, represented by the Close algorithm [6], is based on the theoretical framework introduced in [7] that uses the closure of the Galois connection [8]. In this approach, the frequent closed patterns (and their support) are extracted from the database in a level wise manner. A closed pattern is the greatest pattern common to a set of objects of the database; each non-closed pattern has the same properties (same set of objects containing it, and thus an identical support) as its closure, i.e. the smallest closed pattern containing it. Then, all frequent patterns as well as their support are derived from the frequent closed patterns and their support without accessing the database. Experiments have shown that this approach is much more efficient than the two previous ones on such data.

The fourth approach represented in [9],is based on pattern counting inference. This method relies on the concept of key patterns, where a key pattern is a minimal pattern of equivalence class gathering all pattern common to the same objects of the database relation. Hence, all patterns in an equivalence have the same support and the supports of the non-key pattern of an equivalence class can be determined using the supports of the key patterns of this class. With pattern counting inference, only the

supports of the frequent key patterns (and some infrequent ones) are determined from the database, while supports of the frequent non-key patterns are derived from those of the frequent key patterns. Problem Statement and motivation.

Current methods for mining frequent sets still suffer from the following problems:

• Large candidate set
• Lots of database scan

Our goal is to discover the methods to improve and simplify the same.

III. PROPOSED METHODOLOGY

We propose a new and improved approach by combining two approaches i.e. bottom up counting inference and top down intersection method for generating the frequent sets efficiently in less time.

Proposed Algorithm:

Step 1: In the first step we have converted our large database in compact form through a novel data structure, compact transactional tree to generate a compact database. The advantage of this step is that by doing this the amount of disk space required as well as the total running time can be decreased dramatically on large real world databases. In this the tree based database is divided in two parts: head and body. The head part consists of the support count of each transaction and are ordered in frequency decreasing manner. For the step 1 algorithm 3 is used.

A number of transactions in a large transactional database may contain the same set of items .In this method we have compressed the database by making each unique transaction of the original database to contain only one entry in the corresponding compact database, with a count number recording the number of its occurrence in the original data base. This would allow us to avoid repeatedly scanning the same transaction in the original database.

Step 2 (a) : In the second step, we find frequent patterns through the pattern counting inference method .The pattern counting inference method relies on the concept of key patterns. A key pattern is a minimal pattern of an equivalence class gathering all patterns that have the same objects. The pattern counting inference allows to determine the support of some frequent and infrequent patterns (key patterns) in the database only. The support of all other frequent patterns are derived from the frequent key patterns. This allows to reduce at each database pass, the no. of patterns considered and even more importance to reduce the number of passes in total. This optimization is valid since key patterns have property that all subsets of a key pattern are key patterns and all supersets of a non-key pattern are non-key patterns. The counting inference is performed in a level wise manner if a candidate pattern of size k which support has to be determined is a non key pattern, then its support is equal to the minimal support among the patterns of size k-1 that are its subsets. The important difference in comparison to previous methodology is to determine as much support counts as possible without accessing the database by information gathered in previous passes. For the step 2 (a) algorithm 1 and 2 is used.

Step 2 (b) : Then a top down intersection method is applied in which we intersect the candidate item sets which are infrequent to get the candidate patterns which may be frequent. In order to construct k-itemsets, infrequent (k-1)-itemsets are used. Their union is formed and for their support count and intersection operation is employed between the TIDs of the itemsets. Itemset {CDIJ} and itemset {BCDI} are infrequent. The union of these two itemsets and intersection principles is used to find if the resultant itemset is frequent as follows:

$$T(CDIJ) \cap T(BCDI) = T(CDI) \text{ (which is frequent)}$$

If the result is greater than minimum support, it will be joined to frequent itemsets. If the result is lower than minimum support, it will be pruned off.

Algorithm 1 Pascal (bottom up counting inference)

Notations used in PASCAL

K is the counter which indicates the current iteration. In the Kth iteration, all frequent K patterns and all key patterns among them are determined.

P_K contains after the Kth iteration all frequent kpatterns P together with their support P. sup, and a boolean variable . key indicating if P is a (candidate) key pattern.

C_k stores the candidate k-patterns together with their support (if known), the Boolean variable P.key, and a counter P. pred_supp which stores the minimum of the supports of all (k -1)-sub-patterns of P.

1) φ . sup <—1; φ .key <—true;
2) P0 <- {φ };
3) P1 <-{frequent 1-patterns}; for all p P1 do begin
4) p.pred_supp <— 1; p.key <— (p.supp ≠ 1);
6) end;
7) for (k = 2; Pk-1 ≠ φ ; k + +) do begin
8) Ck <- PASCAL-GEN(PK-1);
9) if c Ck where c.key then
10) forall o D do begin
11) Co <— subset(Ck,o);
12) forall c Co where c.key do
13) c. sup ++;
14) end;
15) forall c Ck do
16) if c. supp ≥ minsup then begin
17) if c.key and c. supp = c.pred_supp then
18) c.key <— false;
19) Pk <- Pk U {c};
20) end;
21) end;
22) return Uk Pk .

Algorithm 2 PASCAL-GEN

Input: Pk-1, the set of frequent (k-1)-patterns p with their support p.supp and the p.key flag.

Output: Ck , the set of candidate k-patterns c each with the flag c.key, the value c.pred_supp, and the support c.supp if c is not a key pattern.

1) insert into Ck select p.item1, p.item2, ..., p.itemk-1, q.item k-1 from Pk-1 p, Pk-1 q where p.item1 = q.item1, ..., p.item k-2 =q.item k-2, p.item k-1 ‹q.item k-1;
2) forall c Ck do begin
3) c.key <— true; c.pred_sup <—+∞ ;
4) forall (k-1)-subsets s of c do begin
5) if s P k-1then

6) delete c from Ck;
7) else begin
8) c.pred_supp <— min(c.pred_supp, s. supp);
9) if not s.key then c.key <— false;
10) end;
11) end;
12) if not c.key then c. supp <— c.pred_supp;
13) end;
14) return Ck.

Algorithm 3 for Compact Database Generator
Input: Original transaction database TDB.
Output: Compact transaction database CDB.
1:root[CTtree] ← ROOT
2: list[item][count] ← null
3: for each transaction Tn in TDB do
4: To ← sort items of Tn in lexicographic order
5: insert(To, CTtree)
6: end for
7: if CTtree is not empty then
8: list ← sort list[item][count] in count descending order
9: for each item i in list[item] do
10: CDB ← write i
11: CDB ← write count[list[i]]
12: end for
13: startNode ← child[root[CTtree]]
14: write(startNode, CDB)
15: else
16: output ”The original transaction database is empty!”
17: end if

procedure insert(T, CTtree)
1: thisNode ← root[CTtree]
2: for each item i in transaction T do
3: if i is not in list[item] then
4: list[item] ← add i
5: end if
6: list[count[i]] ← list[count[i]] + 1
7: nextNode ← child[thisNode]
8: while nextNode = null and item[nextNode] = i do
9: nextNode ← sibling[nextNode]
10: end while
11: if nextNode = null then
12: item[newNode] ← i
13: if i is the last item in T then
14: count[newNode] ← 1
15: else
16: count[newNode] ← 0
17: end if
18: parent[newNode] ← thisNode
19: sibling[newNode] ← child[thisNode]
20: child[newNode] ← null
21: child[thisNode] ← newNode
22: thisNode ← newNode
23: else
24: if item i is the last item in T then
25: count[thisNode]++
26: else
27: thisNode ← nextNode
28: end if
29: end if
30: end for

procedure write(node, CDB)
1: if count[node] = 0 then
2: count[newTrans] ← count[node]
3: nextNode ← node
4: while nextNode = root[CTtree] do
5: newTrans ← insert item[nextNode]
6: nextNode ← parent[nextNode]
7: end while
8: if newTrans is not empty then
9: newTrans ← sort newTrans in list order
10: CDB ← write newTrans
11: end if
12: end if
13: if child[node] = null then
14: write(child[node], CDB)
15: end if
16: if sibling[node] = null then
17: write(sibling[node], CDB)
18: end if

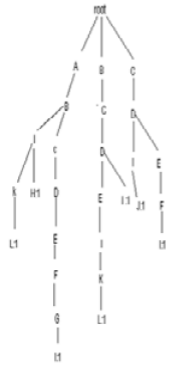
An example of the Proposed Algorithm:

Table I : Transaction database

TID	List of items
001	A,B,C,D,E,F,G,I
002	B,C,D,E,I,K,L
003	A,B,I,K,L
004	A,B,I,H
005	C,D,I,J
006	B,C,D,I
007	C,D,E,F,I

TREE

First step: tree based database



HEAD

Item	I	B	C	D	A	E	K	L	F	G	H	J
Count	7	5	5	5	3	3	2	2	2	1	1	1

BODY

COUNT	LIST OF ITEMS
1	IBCDAEFG
1	IBCDEKL
1	IBAKL
1	IBAH
1	ICDJ
1	IBCD
1	ICDEF

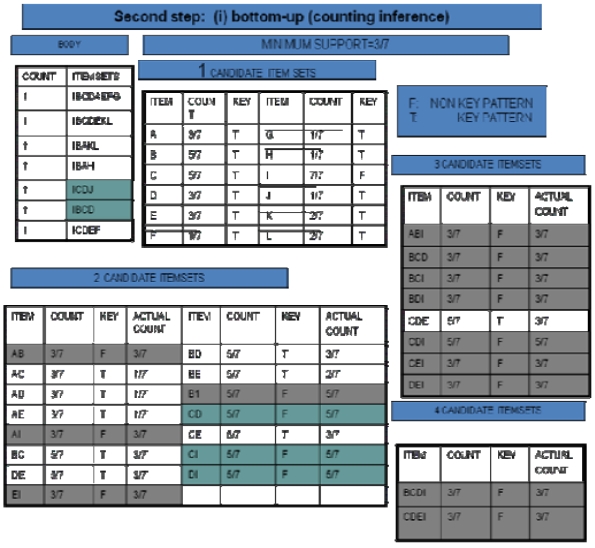


Fig.1. Example of the Proposed Algorithm with bottom up approach

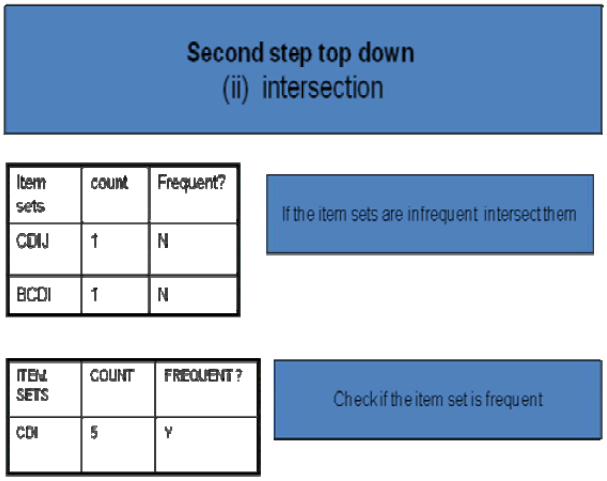


Fig.2. Example of the Proposed Algorithm with intersection

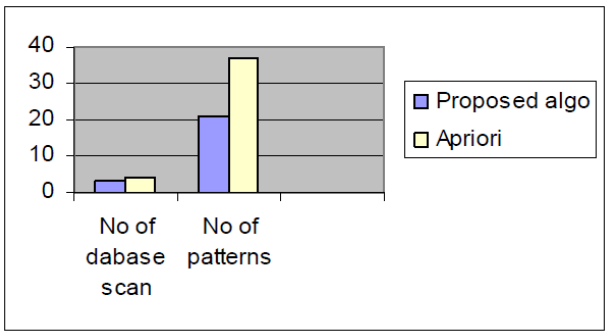


Fig.3. Apriori Vs Proposed algorithm

IV. Conclusion

In this paper, we have integrated the techniques for mining frequent item set to obtain a more efficient algorithm for finding frequent patterns. Efficiency will be in the form of less memory requirement and less computing complexity. We have compressed the database which would allow us to avoid repeatedly scanning the same transaction in the original database. Then we have applied two approaches i.e. counting inference and intersection which would allow less number of databases scans (so that the computation is faster)and thus the mining time required could be decreased. Experimental results show the validity and efficiency of our proposed method. If we apply the proposed algorithm on the example only 3 database passes and only 21 patterns are generated that are to be analyzed, whereas on applying apriori algorithm 4 database passes and 37 patterns are generated that are to be analyzed.

References

[1] Stanchev P., Using Image Mining For Image Retrieval, IASTED International Conference" Computer Science and Technology", Mayb19-21, 2003, cacun, Mexico, 214-218.

[2] Mannila H. and Toivonen H. Levelwise search and borders of theories in knowledge discovery. Technical Report TR C-1997-8, Dept. of Computer Science, U. of Helsinki, Jan .1997.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In Proc. 20th VLDB, Sept. 1994.

[4] Mannila H.and Toivonen H., and Verkamo A.Improved methods for finding association rules. In Proc. AAAI Workshop on knowledge Discovery, pages 181-192, july 1994.

[5] R. J. Bayardo. Efficiently mining long patterns from databases. Proc. SIGMOD conf., pp 85- 93, June 1998.

[6] Pasquier N., Taouil R., and Lakhal L.Efficient mining of Association rules using closed itemset lattices.Information System, vol.24(1):25-46, March 1999.

[7] Pasquier N., Bastide Y., Taouil R., and Lakhal L. Pruning Closed itemset lattices for association rules. Proc. BDA conf., pages 177-196, October 1998.

[8] Ganter B. and wille R. Formal Concept Analysis:Mathematical foundations. Springer, 1999.

[9] Bastide Y., Taouil R., Pasquier N., Stumme G., and Lakhal L. Mining frequent patterns with counting inference. ACM SIGKDD Explorations Newsletter, 2(2):66-75, December 2000.

Author's Profile

Anju Singh
BUIT, BU, Bhopal
asingh0123@rediffmail.com

R. C. Jain
SATI, Vidisha
dr.jain.rcd@gmail.com