International Journal of Information Technology and Knowledge Management
July-December 2010, Volume 2, No. 2, pp. 623-626

# Efficient Mining of Maximal Frequent Concepts

Geetha M[1] & R. J. D'Souza[2]

A frequent itemset is maximal if none of its supersets is frequent. In this paper we propose a method for mining Maximal Frequent Concepts using Quantitative Extended Concept Lattice for discovering maximal frequent concepts from databases which requires only one scan of the database. In mining association rules, the most time consuming job is to discover all frequent itemsets from a large database with respect to a given minimum support. Many sequential and parallel algorithms have been proposed to solve this problem. The most common sequential algorithms are Apriori and its variations. Apriori like algorithms employ a strict bottom-up, breadth-first search and enumerate every frequent itemsets. They require multiple passes over the database. The algorithm Basis Tree finds frequent concepts in a single scan of the database. But proposed algorithm is time efficient when compared to Basis tree algorithm.
Keywords: Concept, Concept Indicator, Distributed, Global Mining, QECL, Support.

## 1. Introduction

The algorithm Apriori [Agrawal et al 1994] is the most well known serial algorithm for mining frequent itemsets. It relies on the Apriori-gen function to generate the candidate itemsets at each iteration. Apriori like algorithms employ a strict bottom-up, breadth-first search and enumerate every frequent itemset. They require multiple passes over the database. The Count Distribution is a parallelized version of Apriori for parallel mining [Agrawal et al 1996]. In this algorithm each processor is responsible for computing the local support counts of all the candidates, which are the support counts in its partition. By exchanging the local support count, all processors then compute the global support counts of the candidates from all partitions. In this approach, every processor is required to keep the local support count of all the candidates at each iteration. One problem in this is the space required to maintain the local support count of a large number of candidate sets. The algorithm Basis tree[ Geetha et al 2007] discovers all the frequent concepts. This algorithm is sequential one and does not work very well with large databases.

In this paper a novel method is proposed based on concept lattice called Mining of Maximal Frequent Concepts using Quantitative Extended Concept Lattice (QECL) for discovering maximal frequent concepts from large databases, which requires only one scan of the database.

[1]Department of Computer Science, M.I.T., Manipal, INDIA

[2]Department of Mathematical & Computational Sciences, N.I.T.K, Surathkal, INDIA.

Email: [1]maiya_geetha@yahoo.com, [2]rjd@nitk.ac.in

## 2. Structure of a Node in QECL

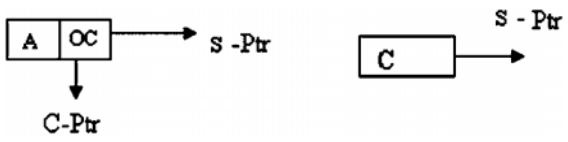


Fig. 1: (a)Node Structure (b) Concept Indicator

There exist two types of nodes as shown in Figure 1(a) and (b).

They are (i) Node structure and (ii) Concept Indicator

(i) Node Structure:

Attribute name(A): The Attribute name represents item number or item name in the object of the Context.

Object Count(OC): Object count represents number of objects in which the attribute or item appears.

Sibling and Child Pointers: Sibling pointer (S-ptr) from a node points to either its sibling or to a Concept Indicator.

(ii) Concept Indicator

The Concept Indicator(C), which indicates that the path traced from the root to the node previous to it represents a concept.

## 3. Methodology Used

The algorithm works in two phases; the local mining phase and global mining phase.

The algorithm uses prefix tree structure to store and count the concepts with different lengths during local as well as global mining phases. To maintain this tree as small as possible, after one pass of the database local QECLs are reduced to contain only frequent concepts based on user

defined minimum support. During the joining phase of Global mining, each concept, say cj, is checked for its frequency. If it is frequent, and if there does not exist any superset containing cj, then it is added to the Global QECL constructed so far. At the same time, all its proper subsets with concept indicators are deleted from the tree. If cj is contained in a frequent superset, then it is not added to the tree; since it will not become maximal frequent.

### 3.1. Construction of Quantitative Extended Concept Lattice

If p is the total number of processors, the original database is initially partitioned into p parts of equal size; each one is assigned to a different processor, and stored in its local disk.

(i) Local Mining Phase: Initially, the QECL has only the root node whose sibling and child pointers are null. The concepts are inserted one by one. For each object in a Context, there exist three concepts namely (i) an empty set (ii) a set containing all attributes of that object and (iii) a set containing all attributes in the dataset. The QECL which is being constructed consists of an empty set as a root node, and a set containing all attributes of an object as one of its branch. Further, the sibling pointer of the last attribute of an object in that concept is pointed to a node containing a character 'C' to indicate that the pattern from the root to this node represents a concept. For each object of the Context, a new branch is created, if the prefix $T_b$ of the object read is not present as the prefix of any branch in the existing QECL. Further, this added branch is marked as concept. If the prefix $T_b$ of the transaction read is matching partially or completely from the beginning of the object, with any branch in the QECL, then the matching prefix or part of the transaction is put into the existing branch by incrementing the corresponding count field value of the nodes in the QECL. Also it is marked as a new concept. Further, a new branch is created for the suffix of the object and is marked as a new concept. The first node of this newly created concept becomes the child of the last node in the matching branch of the existing QECL, depending on whether the prefix matches completely or partially from the beginning of the branch. In both of the above cases, intersection of the newly discovered concept with existing concepts in the tree is obtained. If it results in a non empty intersection and is not matching with the prefixes of any of the branches of the existing QECL, then it is added as a new concept and it becomes the child of the root node in the tree.

(ii) Global Mining Phase (Joining Phase): Let $(|A_1|, B_1)$ and $(|A_2|, B_2)$ be any two quantitative extended concepts. The join of two concepts is defined as follows.

1. If $B_1 \subseteq B_2$ then joining results in two new concepts is as shown below.

   (i) $(|A_1| + |A_2|, B_1 \cap B_2)$

   (ii) $(|A_2|, B_2)$

2. If $B_2 \subseteq B_1$ then joining results in two new concepts is as shown below.

   (i) $(|A_1| + |A_2|, B_1 \cap B_2)$

   (ii) $(|A_1|, B_1)$

3. If $B_1 \cap B_2 = \phi$ then both $(|A_1|, B_1)$ and $(|A_2|, B_2)$ are considered as new concepts.

Let $C_1$, $C_2$, $C_3$,..... $C_n$ be the Quantitative Extended Concept Lattices constructed during local mining phase of this algorithm. These trees are joined in n – 1 iterations to construct a Global QECL by joining two concepts at a time using definition of join defined above. During this iteration, pruning of concepts which are not maximal frequent is done, and finally the tree is kept with only maximal frequent concepts.

### 3.2. QECL Algorithm

The given database is divided into n partitions D1, D2, ….Dn and then distributed among n nodes. Let n = Number of processors to be used to construct Quantitative Extended Concept Lattice. N = Total number of transactions in D.

A. Construction of QECL at node i (Local Mining Phase)

```
Input: Partition Di, i =1, 2, 3…….n
Output: QECL at node i.
Let C be the Global QECL with only root node.
for each node i = 1 to n do
begin
 Construct_qecl(Di, Ci)
 Reduce_qecl(Ci)
end
```

B. Global Mining Phase

```
for each Ci , i = 1 to n do
begin
 Join_qecl(C, Ci)
end
```

### C. Mining Global QECL to Discover Maximal Frequent Concepts

```
for every branch from the root to leaf in Global QECL
C do
begin
 output only those concepts which are marked as
concepts.
end
 /* Function to construct QECL*/
Construct_qecl(Di, Ci)
{ for each object g in Di do
 begin
 Intersect ( g, Ci)
 end }
/* Function to find intersection of concepts*/
Intersect( g, Ci)
{ find the intersection of attributes of g with existing
branches of Ci.
if there exists a non empty intersection Ij
   if Ij is not present in Ci as a beginning part of any
existing branch of
    QECL, then a new branch is created for Ij ,
 increment Ij 's count
 add Ij to the tree as a new concept
   else
 if Ij is present in Ci in the beginning part of any existing
branch of Ci ,
if Ij is not marked as a concept , then
 mark Ij as new concept and increment its count value.
 return; }
/*Function to check frequency of the concept */
Checkfreq_supconcept(Ij, Ci)
{ if there exists any concept containing Ij in the tree Ci
then Ij is not added to Ci
else
{new branch is created for Ij
 increment its count value
    add Ij to the tree as a new concept
    Prune_subsets(min_sup, Ij, Ci) }
return ; }
/* Function to prune infrequent concepts */
Prune_subsets(min_sup, Ii, Ci)
{ if count(Ii) ≥ min_sup
 demark all its proper concepts in Ci
return;}
/* Function to Reduce QECL */
Reduce_qecl(Ci)
{ for every item I in Ci do
begin
if count[I] < min_sup
 then delete I from Ci
end
 return; }
/* Function to join QECLs */
Join_qecl(C, Ci)
 { for every concept (|A2|, B2) in Ci do
begin
for every concept (|A1|, B1) in C do
begin
 if B1 ⊆ B2
 if B1 ∩ B2 is concept and is marked in B1
    then increment count of B1.
 else
 Checkfreq_supconcept((|A1|+|A2|, B1 ∩ B2), C)
 Checkfreq_supconcept ( (|A2|, B2 ), C)
 else if B2 ⊆ B1
    if B1 ∩ B2 is concept and is marked in B1
          then increment count of B1.
 else
 Checkfreq_supconcept((|A1|+|A2|, B1 ∩ B2), C)
 Checkfreq_supconcept ( (|A1|, B1 ), C)
 else if B1 ∩ B2 = ϕ
 Checkfreq_supconcept((|A1|, B1), C)
 Checkfreq_supconcept ( (|A2|, B2 ), C)
 end end return; }
```

## 4. PERFORMANCE ANALYSIS

Experiments are conducted to assess the performance of this algorithm. The algorithm is implemented and used for discovering maximal frequent concepts for data sets containing objects 1K, 3K, 9K, 12K and with 20 attributes by dividing data sets across 2 processors, each having a local memory and local disk. The communication between theses processors is captured via the Message Passing Interface (MPI). The program written in C++ is compiled and linked using the MPI library. The resulting object file is loaded in the local memory of every processor taking part in the computation. The time required to discover maximal frequent concepts using this algorithm against Basis Tree algorithm is shown Table 1.

Table1
Execution Time for QECL and Basis Tree in Seconds

| Data sets | QECL | Basis Tree |
|---|---|---|
| 1K | 11 | 16 |
| 3K | 23 | 34 |
| 9K | 57 | 127 |
| 12K | 112 | 198 |

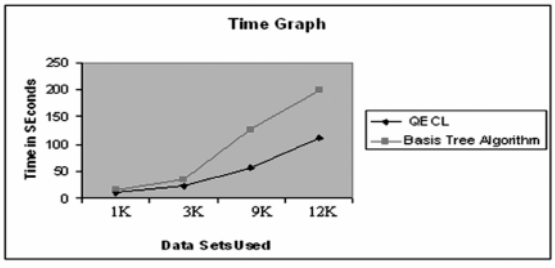


Fig. 2: Execution Time Graph for QECL and Basis Tree

## 4. CONCLUSION

In this paper we proposed an algorithm for mining maximal frequent concepts in a distributive manner. This algorithm uses the concept of quantitative extended lattice and found to be efficient when compared to Basis tree. However, our algorithm QECL can be fine tuned further by considering very large data sets and can be compared with other existing algorithm.

## REFERENCES

[1] Agrawal Rakesh, T. Imielinski, and A. Swami.(1993) "Mining Association Rules between Sets of Items in Massive Databases", In Proceedings of the ACM-SIGMOD International Conference on Management of Data, Washington, D.C, 207-216.

[2] Agrawal, R, and Shafer, J. C (1996) "Parallel Mining of Association Rules", IEEE Transactions on Knowledge and Data Engineering, 8, No. 6, 962-969.

[3] Bayardo R. J (1998) "Efficiently Mining Long Patterns from Databases", In Proceedings of ACM SIGMOD International Conference on Management of Data. Seattle, WA. 85-93.

[4] Geetha M, R.. J. D' Souza "Basis Tree Algorithm to Find Frequent Itemsets" International Conference on Intelligent Systems and Networks" Jagadhri, Haryana, India.

[5] Lin D-I and Z.M. Kedem (1998) "Pincer-Search: A New Algorithm for Discovering the Maximum Frequent Set", In the 6th International Conference on Extending Data base Technology, 105-119.

[6] Patrick Njiwoua, Engelbert Mephu Ngurfo (1997) "A Parallel Algorithm to Build Concept Lattice", In Proceedings of 4th Groningrn International Information Technology Conference for Students, 103-107.