Cluster Based Partition Approach for Mining Frequent Itemsets

[1]Akhilesh Tiwari, [1]Rajendra K. Gupta, and [2]Dev Prakash Agrawal
1Madhav Institute of Technology & Science, Gwalior, India
2Union Public Service Commission, New Delhi, India

Summary
Data Mining is the process of extracting interesting and previously unknown patterns and correlations form huge data stored in databases. Association rule mining- a descriptive mining technique of data mining, is the process of discovering items or literals which tend to occur together in transactions. As the data to be mined is large, the time taken for accessing data is considerable. This paper describes a new approach for association mining, based on Master-Slave architecture. It uses hybrid approach – a combination of bottom up and top down approaches for searching frequent itemsets. The Apriori algorithm performs well only when the frequent itemsets are short. Algorithms with top down approach are suitable for long frequent itemsets. This new master slave architecture based algorithm combines both bottom-up and top-down approach. The Prime number based representation consumes less memory as each transaction is replaced with the product of the assigned prime numbers of their items. It reduces the time taken to determine the support count of the itemsets. The Prime number based representation offers the flexibility for testing the validity of metarules and provides reduction in the data complexity.

Key words:
Frequent patterns, Candidate distribution, Hybrid approach, KDD

1. Introduction

Due to widespread computerization and affordable storage facilities, an enormous wealth of information is embedded in huge databases belonging to different enterprises. Such databases, whether their origin is the business enterprise or scientific experiment, have spurred a tremendous interest in the areas of Knowledge Discovery and Data Mining [1]-[2]. These areas have motivated allowed statisticians and data miners to develop faster analysis tools that can help sift and analyze the stockpiles of data, turning up valuable and often surprising information.
Data mining is the act of drilling through huge volumes of data to discover relationships, or answer queries too generalized for traditional query tools. Data mining is part of the process known as Knowledge Discovery in Databases (KDD) [1]-[3], which is the automated approach of the extraction of implicit, understandable, previously unknown and potentially useful information from large databases. For extraction of such valuable information, the KDD process follows an iterative sequence of steps that include data selection and integration, data cleaning and preprocessing, data mining and algorithm selection, and, finally, post processing and knowledge presentation.

2. Association Rule Problem

The problem of mining association rules is to generate all rules that have support and confidence greater than or equal to some user specified minimum support and minimum confidence threshold respectively.
Association rule mining involves detecting items, which tend to occur together in transactions, and the association rules that relate them [4]-[5]. Mining frequent itemsets is a fundamental and essential operation in data mining applications including discovery of association rule, strong rules, correlations, sequential rules and episodes. Due to the huge size of data and amount of computation involved in data mining, high-performance computing is an essential component for any successful large-scale data mining applications.
Association rule mining finds the set of all subsets of items that frequently occur in many database records or transactions, and additionally extracts rule on how a subset of items influences the presence of another subset. Consider I= $\{i_1, i_2, \ldots\ldots i_m\}$ as a set of items. Let D, the task relevant data, is a set of database transactions where each transaction T is a set of items such that T is a subset of I. Each transaction is associated with an identifier, called TID. Let A be a set of items. A transaction T is said to contain A if and only if A is a subset of T.

An association rule is an implication of the form A => B, where A and B are subsets of I and A∩B is also a subset of I. The rule A => 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 the probability, P (AUB). The rule A => 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,

Support (AUB) = P (AUB)
Confidence (AUB) = P (B/A)

Manuscript received June 5, 2009
Manuscript revised June 20, 2009

The definition of a frequent pattern relies on the following considerations. A set of items is referred to as an itemset (pattern). An itemset that contains K items is a K-itemset. The set {X,Y} is a 2-itemset. The occurrence frequency of an itemset is the number of transaction that contain the itemset. This is also known as the frequency or the support count of an itemset. An itemset satisfies minimum support if the occurrence frequency of the itemset is greater than or equal to the minimal support threshold value defined by the user [7]. The number of transaction required for the itemset to satisfy minimum support is therefore referred to as the minimum support count. If an itemset satisfies minimum support, then it is a frequent itemset.
A frequent itemset is called closed if it does not have any superset with the same support [5]. A frequent itemset is said to be maximal if it has no supersets that are frequent. The collection of maximal frequent itemsets is a subset of the collection of closed frequent itemsets, which is a subset of the collection of all frequent itemsets. Maximal frequent itemsets are necessary for generating association rules.

The problem of mining association rules could be decomposed into two sub problems:

1. Find out all large itemsets and their support counts. A large itemset is a set of items which are contained in a sufficiently large number of transactions, with respect to a support threshold minimum support.

2. From the set of large itemsets found, find out all the association rules that have a confidence value exceeding a confidence threshold minimum confidence.

Since the solution of the second subproblem is straightforward, here we are concentrating only on the first subproblem.

3. Basic Association Rule Mining Algorithms

3.1 Apriori algorithm

The Apriori algorithm[4]-[5]-[6] is also called the level-wise algorithm and was proposed by Agrawal and Srikanth in 1994. It is the most popular algorithm to find all of the frequent sets which uses the downward closure property. The advantage of the algorithm is that before reading the database at every level, it prunes many of the sets which are unlikely to be frequent sets by using the Apriori property, which states that all nonempty subsets of frequent sets must also be frequent. This property belongs to a special category of properties called anti-monotone in the sense that if a set cannot pass a test, all of its supersets will fail the same test as well.
Using the downward closure property and the Apriori property, this algorithm works as follows. The first pass of the algorithm counts the number of single item occurrences to determine the L_1 or single member frequent itemsets. Each subsequent pass, K, consists of two phases. First, the frequent itemsets L_{k-1} found in the (k-1)th pass are used to generate the candidate itemsets C_k, using the Apriori candidate generation algorithm. Next, the database is scanned and the support of the candidates in C_k is determined to ensure that C_k itemsets are frequent itemsets.

Candidate Generation Algorithm

The candidate generation procedure works as follows. Suppose that the set of frequent 3-itemsets, L_3, are {1,2,3}, {1,2,5},{1,3,5},{2,3,5},{2,3,4}. The 4-itemsets that are generated as candidate itemsets are the supersets of these 3-itemsets and are {1,2,3,5}, {2,3,4,5}, which satisfy the downward closure property. More formally, if k is the pass number, L_{k-1} is the set of all frequent (k-1)-itemsets, C_k is the set of candidate sets of pass k, the candidate generation procedure is as follows:

gen_candidate_itemsets with the given L_{k-1} as follows:

$C_k = \Phi$

for all itemsets $l_1 \epsilon L_{k-1}$ *do*

for all itemsets $l_2 \epsilon L_{k-1}$ *do*
if $l_1[1] = l_2[1] \wedge l_1[2] = l_2[2] \wedge \ldots \wedge l_1[k-1] < l_2[k-1]$

then $c = l_1[1], l_1[2] \ldots l_1[k-1], l_2[k-1]$

$C_k = C_k \cup \{c\}$

So once candidate sets are generated those sets are subject to pruning process to ensure that all the subsets of the candidate set are already known to be frequent itemsets.

Pruning Algorithm

The pruning step eliminates some candidate sets which are not found to be frequent, and is:

Prune(C_k)

for all $c \in C_k$

for all (k-1)-subsets d of c do

if d ∉ L_{k-1}
then $C_k = C_k - \{c\}$

3.2 Apriori algorithm description

The Apriori frequent itemset discovery algorithm uses the above algorithms (candidate generation and pruning) at every iteration. It goes from level 1 to level k or until no candidate set remains after pruning. The Apriori algorithm is as follows.

Initialize: K: = 1, C_1 = all the 1 – itemsets;
Read the database to count the support of C_1 to determine L_1.

L_1 := {Frequent 1- itemsets};
K:= 2; // K represents the pass number //
While (K-1 ≠ Null set) do
begin

C_k := gen_candidate_itemsets with the given L_{k-1}

Prune (C_k)
for all transactions t ∈ T do
Calculate the support values;
L_k := All candidates in C_k with a minimum support;
K:= K+1;
End
Answer := $\cup_k L_k$;

For an example of the Apriori algorithm, suppose the following transaction database is given below:

Table1: Transaction Database

	A1	*A2*	*A3*	*A4*	*A5*
Ψ t1	1	0	0	0	1
Ψ t2	0	1	0	1	0
Ψ t3	0	0	0	1	1
Ψ t4	0	1	1	0	0
Ψ t5	0	0	0	0	1

Suppose σ_{min} = 20 %, which means that an itemset must supported by at least one transaction to be frequent because T only has five records.
In the first pass, where k=1, T is read to find the support of 1- itemsets given below.

{1}→ 1, {2}→2, {3} →1, {4} → 2, {5} → 3

L_1 := { {1}→ 1, {2}→2, {3} →1, {4} → 2, {5} → 3}

In the second pass where k=2 , the candidate set C_2 becomes

C_2 := { {1,2}, {1,3}, {1,4}, {1,5}, {2,3}, {2,4},{2,5},{3,4},{3,5},{4,5} }

The pruning step does not change C_2 as all subsets are present in C_1.
Read the database to count the support of elements in C_2 to get:

{ {1,2}→ 0, {1,3}→0, {1,4} →0, {1,5}→1, {2,3}→ 1, {2,4} →1, {2,5}→0, {3,4}→0, {3,5}→0, {4,5}→1} and reduces to

L_2 = { {1,5}→1, {2,3}→1, {2,4}→1, {4,5}→1}

In the third pass where k=3, the candidate generation step proceeds by:

In the candidate generation step,

- Using {1,5} and {4,5} it generates {1,4,5}
- Using {2,3} and {2,4} it generates {2,3,4}
- Using {2,4} and {4,5} it generates {2,4,5}

So C_3 := { {1,4,5}, {2,3,4}, {2,4,5}}
The pruning step prunes {1,4,5}, {2,3,4},{2,4,5} as not all subsets of size 2, i.e., {1,4}, {3,4}, {2,5} are not present in L_3.
So C_3 := Φ

The total frequent sets become L:= $L_1 \cup L_2$.

4. Partition Algorithm

The partition algorithm [1]-[2]-[3] is based in the observation that the frequent sets are normally very few in number compared to the set of all itemsets. As the result, if the set of transactions are partitioned in to smaller segments such that each segment can be accommodated in the main memory, then the set of frequent sets of each of these partitions can be computed. Therefore this way of finding the frequent sets by partitioning the database may improve the performance of finding large itemsets in several ways:

- By taking advantage of the large itemset property, this is that a large itemset must be large in at least one of the partitions. This idea can help to design algorithms more efficiently than those based on looking at the entire database.
- Partitioning algorithms may be able to adapt better to limited main memory. Each partition can be created such that it fits in to main memory. In addition it would be expected that the number of itemsets to be counted per partition would be smaller than those needed for the entire database.
- By using partitioning, cluster based and/or distributed algorithms can be easily created, where each partitioning could be handled by a separate machine.
- Incremental generation of association rules may be easier to perform by treating the current state of the database as one partition and treating the new entries as a second partition.

In order to achieve all the above advantages of partitioning the transaction database, the partition algorithm works as follows:

The partition algorithm uses two scans of the database to discover all frequent sets. In one scan, it generates a set of all potential frequent itemsets by scanning the database. This set is a superset of all frequent itemsets, i.e. it may contain false positives, but no false negatives are reported. During the second scan, counters for each of these itemsets are setup and their actual support is measured in one scan of the database.

The partition approach of generating frequent itemsets is given below:

P = partition_database (T); N = Number of partitions;
// Phase 1
for i = 1 to n do begin
read _in_partition(T_i in P)
L_i = generate all frequent itemsets of Ti using apriori method in main memory.
End

// Merge Phase
For (k=2 ; $L_i^k \neq \Phi$*, i= 1,2,……,n; k++) do begin*

$C_k^G = Y_{i-1}^n L\ i^k$

End

// Phase II
For i= 1 to n do begin

Read _ in _ partition(T_i *in P)*
For all candidates c $\in C^G$ compute $s(c)T_i$

End

$L^G = \{ c \in C^G / s(c)T_i \geq \sigma \}$

Answer = L^G

As given the partition algorithm above, here is the example of implementing it:

Table2: Transaction Database

	A_1	A_2	A_3	A_4	A_5
Ψ_{t1}	1	0	0	0	1
Ψ_{t2}	0	1	0	1	0
Ψ_{t3}	0	0	0	1	1
Ψ_{t4}	0	1	1	0	0
Ψ_{t5}	0	0	0	0	1
Ψ_{t6}	0	1	1	1	0

Here is the transaction database, A = {A1, A2, A3, A4, A5}, assume σ = 20 %.
Here the database is partitioned in to 3 partitions say ξT_1, ξT_2, ξT_3, each containing 2 transactions. The first partition ξT_1 contains 1 to 2 transactions, ξT_2 contains 3 to 4, and ξT_3 contains 5 to 6 transactions. Here the local support is equal to the given support, which is 20%. So $\sigma = \sigma_1 = \sigma_2 = \sigma_3 = 20\%$.

The working of partition algorithm is as follows:

L_1 := the frequent sets from the partition in ξT_1, which are found using the apriori algorithm on ξT_1 separately.

L_2 := the frequent sets from the partition in ξT_2, which are found using the apriori algorithm on ξT_2 separately.

L_3 := the frequent sets from the partition in ξT_3, which are found using the apriori algorithm on ξT_3 separately.

In phase II, the candidate set as

$C := L_1 \cup L_2 \cup L_3$

Later read the database once again to compute the global support of the sets in C and get the final set of frequent sets.

4.1 Problem specification

In recent years several fast algorithms including Apriori [3] and Partition [2] for generating frequent itemsets have been suggested in the literature[4]-[7]-[8]-[9]-[10] . A critical analysis of these has led the authors to identify the following limitations/shortcomings in them.

- By using partitioning, cluster based parallel and/or distributed algorithms can be easily created, where each partitioning could be handled by a separate machine.

- Most of the algorithms[2]-[8]-[9] for discovering Frequent Patterns require multiple passes over the database resulting in a large number of disk reads and placing a huge burden on the I/O subsystem.

- Algorithms are available for maintaining the association rules[2]-[10] due to addition or deletion of transactions in the database. However, algorithms are not available for mining incremental rules due to addition of more items.

- Currently available Algorithms [1]- [4]-[8] for mining frequent itemsets do not offer flexibility and reusability of computation during mining process.

5. Proposed Algorithm

To address the above-mentioned limitations/shortcomings, a new Algorithm for Mining Frequent Itemsets using cluster based partition approach (CBPA) is being proposed. This algorithm integrates both the bottom up search as well as the top-down search. This algorithm is suitable for itemsets of any size. It uses the top down approach to find the frequent subsets of itemsets. The bottom up approach is used to find the supersets of the frequent itemsets. Most of the algorithms for mining the frequent items are based on bottom-up search approach. In this approach, the search starts from 1 itemsets and extends one level in each pass until all maximal frequent itemsets are found. This approach performs well if the length of the maximal itemset is short. If the maximal itemset is longer, top down search is suitable. For a transaction with a medium sized maximal frequent set, a combination of both these approaches performs well. This algorithm adopts Candidate distribution method to distribute the candidates among all nodes. The support count of the supersets and the subsets are found effectively from the prime number representation method. The Prime number representation reduces the memory needed for storing the items of the transactions by assigning a unique prime number for each item.

Proposed algorithm uses prime numbers to represent the items in the transaction. Each item is assigned an unique prime number. Each transaction is represented by the product of the corresponding prime number of individual items in the transaction. Since the product of the prime number is unique, modulo division of a transaction's prime product by the prime product of the itemset can assure the presence or absence of the itemset in any transaction.

- If the remainder is zero, then the itemset is present in the transaction.

- If the remainder is Non-zero, then the itemset is not present in the transaction.

By checking the presence of itemset in any transaction using the above discussed method / approach, support count can be calculated very quickly. Each transaction in the database can be represented in a single number by using prime representation.

5.1 I llustration-I

Consider a sample database as shown in Table 3. In the proposed approach, every item is assigned a unique prime number as shown in Table 4. Table 5 shows the transaction table with the itemset replaced by the product of the equivalent prime numbers of the itemset.

Table 3: Sample Database

Tid	Transaction
Ψ_{t1}	A,B,C,D
Ψ_{t2}	A,B
Ψ_{t3}	C,D,E,F,G,H
Ψ_{t4}	A,C,F,H
Ψ_{t5}	A,D,E,F,G
Ψ_{t6}	B,H,I
Ψ_{t7}	A,J,K,L,M
Ψ_{t8}	A,D,F,G
Ψ_{t9}	A,C,E,G,H
Ψ_{t10}	C,E,M
Ψ_{t11}	A,C,E
ψ_{t12}	A,B,C

Table 4: Prime Assignments

Items	Prime Number Equivalent
A	2
B	3
C	5
D	7
E	11
F	13
G	17
H	19
I	23
J	29
K	31
L	37
M	41

Table 5: Novel Representation

Tid	Transaction	Transaction multiple
Ψ_{t1}	2*3*5*7	210
Ψ_{t2}	2*3	6
Ψ_{t3}	5*7*11*13*17*19	1616615
Ψ_{t4}	2*5*13*19	2470
Ψ_{t5}	2*7*11*13*17	34034
Ψ_{t6}	3*19*23	1311
Ψ_{t7}	2*29*31*37*41	2727566
Ψ_{t8}	2*7*13*17	3094
Ψ_{t9}	2*5*11*17*19	35530
Ψ_{t10}	5*11*41	2255
Ψ_{t11}	2*5*11	110
ψ_{t12}	2*3*5	30

Support count of an item or an itemset can be easily determined by performing modulo division operation with the transaction's prime product and item's prime or the itemset's prime product. If the modulo operation gives a zero remainder, it indicates that the item or itemset is in the transaction. If the remainder is non-zero, it indicates that the item or itemset is not present in the transaction.

In the illustration, support count of itemset {B,C} can be found by performing the modulo division of each transaction's prime product by the product '55' of item 'B's corresponding prime number '5' and item 'C's corresponding prime number '11' as shown in Table 6.

The support count of itemset {B,C} is found to be '4' as the modulo division operation of the four transactions with 55 gives zero remainder and the modulo operation with other transactions resulted in a Non-zero remainder. This representation reduces the memory needed for storing the items of the transactions by assigning an equivalent prime number for each item.

Table 6: Support count determination for itemset {B,C}

Tid	Modulo Division	Remainder	Item's Presence
Ψ_{t1}	210 mod 55	Non-zero	No
Ψ_{t2}	6 mod 55	Non-zero	No
Ψ_{t3}	1616615 mod 55	0	Yes
Ψ_{t4}	2470 mod 55	Non-zero	No
Ψ_{t5}	34034 mod 55	Non-zero	No
Ψ_{t6}	1311 mod 55	Non-zero	No
Ψ_{t7}	2727566 mod 55	Non-zero	No
Ψ_{t8}	3094 mod 55	Non-zero	No
Ψ_{t9}	35530 mod 55	0	Yes
Ψ_{t10}	2255 mod 55	0	Yes
Ψ_{t11}	110 mod 55	0	Yes
ψ_{t12}	30 mod 55	Non-zero	No

In the first pass the algorithm scans the data set and computes the support count of all 1-itemsets. The infrequent 1-itemsets are removed from further evaluation. Each item is represented by a unique assigned prime number and each transaction is represented by the multiple of the assigned prime number of the items in the itemset. The maximal length itemset M is found and the support count is found for all the transactions with the same length. The support count is found using prime number representation method. The two possibilities are as follows:

- If the support count is greater than or equal to the minimal support count, it is treated as the maximal frequent set and the procedure ends.
- If the support count for the itemsets of length M is less than the minimal support count, the subsets of length equal to N=M/2 is generated and their support count is determined

This again leads to the following possibilities

- If the support count of the sets of size N is greater than the minimal support count, all possible supersets of size N+N/2 are generated and their support count is determined.

- If the support count is less than the minimal support count, the subsets of length equal to N/2 is generated and their support count is determined.

This procedure is repeated in the same manner until the maximal frequent itemset is found. Support count is determined using prime number representation method.

5.2 Algorithm for master

This algorithm is based on Cluster based Master-Slave architecture using candidate distribution technique. Candidate distribution technique reduces the communication overhead between master and slave nodes. Candidate distribution technique assigns the candidate itemsets generated from different parts of database to different processors and each processor is assigned disjoint candidates, independent of other processors. At the same time, the database is shared among all processors, so that each processor can generate global counts independently.

Steps of Algorithm for Master:

1. Find the infrequent itemsets of length 1 and store them in IF 1
2. Remove the infrequent 1-items as denoted by IF1 in all transactions.
3. Assign separate Prime Number Pj to each unique item ITj for n-items.
4. Represent the itemsets in Prime Number Representation form as follows:
   (a) Replace each Transaction's item ITj by Corresponding Prime Number Pj.
   (b) Represent each Transaction Tj of Size m by the multiple Mj of all the prime number representation Pj of the items in the transaction (P1 x P2 x P3 x…….xPm) and store them in shared memory.
5. Find the size Maxlength of maximal size transaction in Database and put it in shared memory.

6. For each node j in the cluster
   Divide the transactions equally based on the number of nodes and assign to j-th Node.

   Connect to j-th node's server program to initiate process.
   End loop
7. For each node j in the cluster
   Wait until result comes from j-th node
   Show the result from j-th node
   End loop

5.3 Algorithm for slaves

Steps of Algorithm for Slaves:

1. Wait until master initiates process.
2. Read Transactions Tj from shared memory.
3. Read Prime number multiple Mj of Transactions from shared memory.
4. Read the size Maxlength of Maximal size transaction from shared memory.

SG=empty
Where SG is the subset group
FrequentItemset = empty
Start = 1

End = Maxlength
j = Maxlength

Exitflg = 0

5. Do While Exitflg = 0
   For each transaction Tj with size >= j do
   For each itemset S of Transaction Tj with size j do
   If S is not in SG AND if IF1's items are not a subset of S then
   Find the Support count of itemset S
   Mj mod K and counting its presence using the remainder and store it in SUPPORT
   If SUPPORT >= minsupport then Add S to FrequentItemset

   End If

   Add S to SG;
   End If
   End loop

   End loop

   Clear the SG

   If FrequentItemset is not empty
   Start = j
   j = Round ((Start + End)/2)
   If j = End then
   Send all Itemset AllFrequentItemset to master
   Exitflg = 1
   Exit the Do loop
   End if
   Find the infrequent items in infrequent j-size Itemsets and add them to IF1
   Add FrequentItemset to AllFrequentItemset
   Clear the FrequentItemset
   Else

   End= j
   j = Round (Start + End) /2
   If j= Start -1 then
   Send all Itemset AllFrequentItemset to master

Exit the Do loop.

End if
End if
End Do loop

The Master node prunes the transactions by removing 1-infrequent itemsets and stores the Prime number multiple for each transaction in shared memory. It finds the maximal length transaction size Maxlength and puts in shared memory. It divides the transactions equally to each node for candidate generation. Though horizontal partitioning, Vertical partitioning methods can be used to divide and distribute the transactions, horizontal partitioning method is adopted, as it demands minimum communication.
If there are S number of slaves and T number of transactions, then T/S number of transactions are assigned to each slave if T is a integral multiple of S. Otherwise, S-1 slaves will be assigned T/S transactions and S^{th} slave will be assigned (T/S + mod (T/S)) transactions. Master connects to each slave node and initiates the process of finding the frequent itemset. Finally, the master node shows the global frequent itemsets after gathering the local frequent itemsets. After the Master node initiates the slave node, it reads the allotted number of transactions and Maximal length transaction size Maxlength.

5.4 Illustration – II

For the set of transactions given in table 1, the master removes the 1-infrequent items {J, K, L} since their support count is less than the minimum support count of 2. The transactions are then represented with their assigned prime numbers and stored in common memory. The transactions are divided equally sent to the slaves. In this illustration,

Total number of transactions T = 12
Number of slaves S = 3
Number of transactions sent to each slave = 4
Slave node 1 will process candidates from 1-4 transactions
Slave node 2 will process candidates from 5-8 transactions
Slave node 3 will process candidates from 9-12 transactions
Maxlength = 6

The transactions after removing the 1-infrequent items are shown in table 7. Table 8 illustrates the Prime number representation of the transactions after pruning the 1-infrequent items.

Table 7: Transactions after pruning

Tid	Transaction
Ψ_{t1}	A,B,C,D
Ψ_{t2}	A,B
Ψ_{t3}	C,D,E,F,G,H
Ψ_{t4}	A,C,F,H
Ψ_{t5}	A,D,E,F,G
Ψ_{t6}	B,H,I
Ψ_{t7}	A,M
Ψ_{t8}	A,D,F,G
Ψ_{t9}	A,C,E,G,H
Ψ_{t10}	C,E,M
Ψ_{t11}	A,C,E
ψ_{t12}	A,B,C

Table 8: Novel Representation after pruning

Tid	Transaction	Transaction multiple
Ψ_{t1}	2*3*5*7	210
Ψ_{t2}	2*3	6
Ψ_{t3}	5*7*11*13*17*19	1616615
Ψ_{t4}	2*5*13*19	2470
Ψ_{t5}	2*7*11*13*17	34034
Ψ_{t6}	3*19*23	1311
Ψ_{t7}	2*41	82
Ψ_{t8}	2*7*13*17	3094
Ψ_{t9}	2*5*11*17*19	35530
Ψ_{t10}	5*11*41	2255
Ψ_{t11}	2*5*11	110
ψ_{t12}	2*3*5	30

Slave node 1 determines the support count of the candidate itemset of length = Maxlength i.e. 6. Since the support count of { C, D, E, F, G, H } is 1, Maxlength = Maxlength/2
Maxlength = 6/2 = 3
It generates the candidate itemsets of length three as {A,B,C}{B,C,D} {C,D,E} {D,E,F}{E,F,G}{F,G,H} {A,C,F} {C,F,H} {A,F,H} {D,G,H} and determines their support count. If no candidate itemset has a support count greater than 2, Maxlength = round (Maxlength / 2), else Maxlength = round (Maxlength +Maxlength /2).
All the nodes proceed in this manner till the maximum frequent itemset is found. Master finally receives all frequent itemset from nodes and displays them.

6. Conclusion

In this paper, a new Algorithm for Mining Frequent Itemsets using Cluster Based Partition Approach was proposed. The innovative Prime number representation stores only one number for each transaction, it may need less memory. The computational complexity is reduced as the product of their assigned prime numbers represents each candidate itemset. The support count of any set is found without any additional scan of the database. The pruning of infrequent items in the first scan reduces the size of the dataset in the main memory. Present algorithm also provide support for the incremental generation of association rules i.e. it exhibits scalability and can be efficiently used to find low support itemsets within the large database.

References

[1] Arun K Pujari. Data Mining Techniques (Edition 5):Hyderabad, India: Universities Press (India) Private Limited, 2003.
[2] Margatet H. Dunham. Data Mining, Introductory and Advanced Topics: Upper Saddle River, New Jersey: Pearson Education Inc., 2003.
[3] Jiawei Han. Data Mining, concepts and Techniques: San Francisco, CA: Morgan Kaufmann Publishers.,2004.
[4] R.K. Gupta. Development of Algorithms for New Association Rule Mining System, Ph.D. Thesis, Submitted to ABV-Indian Institute of information Technology & Management, Gwalior, India, 2004.
[5] A. T. Bjorvand. Object Mining: A Practical Application of Data Mining for the Construction and Maintenance of Software Components. Proceedings of the Second European Symposium, PKDD-98, Nantes, France, 1998, pp :121-129.
[6] Akhilesh Tiwari, R. K. Gupta, D.P. Agrawal, Mining Frequent Itemsets Using Prime Number Based Approach. In Proc. 3rd International Conference on Advanced Computing and Communication Technologies (ICACCT), India, November 08-09,2008, pp: 138-141.
[7] M. Houtsma and A. Swami. Set Oriented Mining for Association Rules in Relational Databases. In Proceedings of 11th International conference on Data Engineering, 1995, pp 25-33,.
[8] Agarwal R., Imielinski T., and Swami A. Mining associations between sets of items in massive databases. In Proceedings of the ACM SIGMOD International Conference on Management of Data, Washington D.C. , May 1993, pp. 207-216.
[9] M. Houtsma and A. Swami, Set Oriented Mining for Association Rules in RelationalDatabases. In Proceedings of 11th IEEE International Conference on Data Engineering, 1995, pp : 25-33.
[10] Rakesh Agrawal and R. Srikant . Fast Algorithm for Mining Association Rules in Large Databases, Proceedings of the 20th International Conference on Very Large Databases, Santigo, Chile, 1994, pp 487-499.