K-Partition Model for Mining Frequent Patterns in Large Databases

Nidhi Sharma
Department of Computer Science & Engineering
BUIT, BU, Bhopal, India
nidhi.sharma1311@gmail.com

Anju Singh
Department of Computer Science & Engineering
BUIT, BU, Bhopal, India
asingh0123@rediffmail.com

Abstract: **Mining frequent patterns has always been a great field of research for investigators. Various algorithms were developed for finding out frequent patterns in an efficient manner. But the major drawback of all these researches is the increased number of database scans. Partition algorithm is one of the approaches for mining frequent patterns but the large number of database scans required in this algorithm makes the mining process slow. Few developments have succeeded in reducing the number of database scans to two. Here an attempt has been made to develop a K-Partition algorithm which requires one database scan. Whole database is compressed in the form of Karnaugh Map, having very small size i.e. a fraction of the whole database. Then partition algorithm can be used to identify frequent patterns using K-Map model. Thus this approach brings efficiency in terms of time taken by processor for mining frequent patterns.**

Keywords: ***Frequent patterns, K-Partition, Karnaugh Map, Database scans.***

I. INTRODUCTION

Data mining is the exploration and analysis of large data sets, in order to discover meaningful patterns and rules. The main objective is to find effective ways to combine the computers power to process data with the human eye ability to detect patterns. Most of the techniques of data mining are designed for, and work best with, large data sets.

The advent of computing technology has significantly influenced our lives and two major impacts of this effect are Business Data Processing and Scientific Computing. During the initial years of the development of computer techniques for business, computer professionals were concerned with designing files to store the data so that information could be efficiently retrieved. There were restrictions on storage size for storing data and on the speed of accessing the data. There has been a lot of research in developing algorithms for mining frequent itemsets in an efficient manner. Most of them enumerate all frequent itemsets [1, 2]. Some algorithms generate closed itemsets and maximal frequent itemsets [3] to achieve efficiency in terms of complexity but they still need to mine whole database, thus these methods are not efficient for mining evolving databases [4,5,6].

Apriori like algorithms, iteratively obtain candidate itemsets of size (k + l) from frequent itemsets of size k. Each iteration require a scan of the original database. It is costly and inefficient to repeatedly scan the database and check a large set of candidates for their occurrence frequencies [7]. Various methods are there to improve Apriori performance [8], but there are still many problems.

One of the approaches is to use Partition algorithm, where in one scan, the set transactions are partitioned into smaller segments such that each segment can be accommodated in main memory. Then, the frequent itemsets can be computed for each of these partitions. This is a super set 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 set up and there actual support is measured in one scan of the database.

In this paper an attempt has been made to reduce the database scan to one by compressing the database in the form of frequency, which will reduce the size of database and then mining is performed on the compressed data set. For this, Karnaugh Map technique is used to store all of the information in a highly compact form and updates easily. Then set theory is used to calculate support count for itemsets.

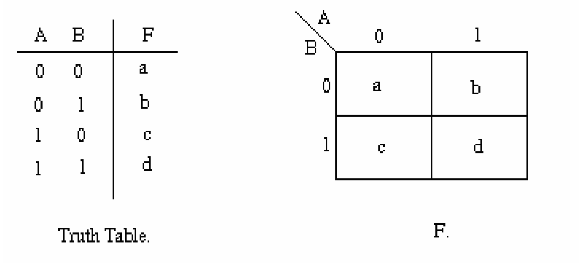
II. LITERATURE SURVEY

A.A-Priori Algorithm: The initial approach, for identifying frequent patterns and hence association rules, is based upon A-Priori algorithm. This algorithm identifies set of frequent items L_k and candidate itemsets C_k for each pass K. Support count of the itemsets can be calculated by scanning the database again and again for each pass, which reduces the performance of algorithm, thus increasing the overall execution time. This is the major drawback of this approach. Various methods are there to improve A-priori performance [8] to some extent but still lot of scope is there for further improvement.

B. Partition Algorithm: Further research has been done in order to improve the efficiency of A-priori algorithm, and hence, Partition algorithm came into existence. Lots of research has already been done using the base of partition algorithm [9,10]. This algorithm logically divides the database into partitions and then for each partition P_i, a set of frequent itemsets L_i is to be identified in one database scan. Then these local frequent itemsets are combined to generate global candidate itemsets. Then the support count is calculated using one more database scan, in order to obtain final set of frequent itemsets.

This approach is improving the A-priori performance but still some scope is there for further improvement.

C. Karnaugh Map Approach: Further researches evolved with a new concept of Karnaugh Map [11]. A Karnaugh map provides a pictorial method of grouping together expressions with common factors and therefore eliminating unwanted variables. The Karnaugh map can also be described as a special arrangement of a truth table. The diagram below illustrates the correspondence between the Karnaugh map and the truth table for the general case of a two variable problem [12].



D. K-Priori Algorithm: Using the concept of K-Map, a new algorithm was developed to reduce the number of database scans to one. In this algorithm K-priori [11], karnaugh map is used to store the database transactions in reduced form which needs only one scan of the database and then A-priori algorithm is used to identify frequent sets. But now, support count can be calculated directly from K-Map, so no further scanning of the database is required.

Here in this paper, we are making an attempt to implement the concept of Karnaugh Map along with Standard Partition Algorithm, so that the number of database scans can be reduced to one.

III. Proposed Algorithm K-Partition

K-Partition algorithm is based on designing Karnaugh Map from existing database, which requires only one scan of the database. Then Partition algorithm can be used for generating frequent patterns. The name K-Partition is given for the combined concept of Partition algorithm and K-Map.

Generation of K-Map:

Let I = {i_1, i_2, i_3,…, i_n} be the set of all items, where each item is a binary variable. It can hold either **0** or **1** at a time. T = {t_1, t_2, t_3,….., t_n} be the set of all transactions. Each transaction t_i contains a subset of items chosen from **I**. In association a collection of 0 or more items is termed as itemset. If an itemset contains k items, it is called K-itemset.

A table K-Map is created with first two bits representing items I1, I2 in the Rows and next two bits representing items I3, I4 in the columns. Then for each transaction in the database we can read the items and can mark 1 in the corresponding row and column of K-Map. Next time if same bits are appearing then its value can be incremented by one otherwise place 1 in the corresponding row and column as shown in table 1. Values present in the K-Map shows the frequencies of the items.

Table 1 K-Map:

I1I2 \ I3 I4	00	01	10	11
00	0	0	0	0
01	0	2	0	1
10	1	0	0	0
11	0	0	1	0

Support Counts can be calculated using the following formulas:

$\text{Supp}(I_k) = \{\Sigma\ a_{ij}: \text{for all } a_{ij}\quad (r_i \text{ U } r_{i+1} \text{ U} \ldots\ldots\ldots r_x)\}$ ………………… (3a)

Where $I_k\quad R_{set}$

$\text{Supp}(I_k) = \{\Sigma\ a_{ij}: \text{for all } a_{ij}\quad (c_i \text{ U } c_{i+1} \text{U} \ldots\ldots\ldots c_y)\}$ ………………….. (3b)

Where $I_k\quad C_{set}$

$\text{Supp}(I_k\ I_{k+1}) = \{\Sigma\ a_{ij}: \text{for all } a_{ij}\quad (r_i\quad r_{i+1}\quad \ldots\ldots)\}$ …………………… (3c)

Where $(I_k\ I_{k+1})\quad R_{set}$

$\text{Supp}(I_k\ I_{k+1}) = \{\Sigma\ a_{ij}: \text{for all } a_{ij}\quad (c_i\quad c_{i+1}\quad \ldots)\}$...................................(3d)

Where $(I_k\ I_{k+1})\quad C_{set}$

Partition Algorithm:

This algorithm executes in two phases. In the first phase, the partition algorithm logically divides the database into a number of non-overlapping partitions. The partitions are considered one at a time and all frequent itemsets for that partition are generated. Thus, if there are *n* partitions, Phase I of the algorithm take *n* iterations. At the end of Phase I, these frequent itemsets are merged to generate a set of all potential frequent itemsets. In this step, the local frequent itemsets of same lengths from all *n* partitions are combined to generate the global candidate itemsets. In Phase II, the actual support count for these itemsets is generated and the frequent itemsets are identified.

Proposed Algorithm:

```
Begin

{    Initialize :  n = number of partitions required
                   N = number of transactions to be
                       analyzed
                   m = N/n   // number of transactions
                                in each partition//
// Phase I
    for i = 1 to n do begin
        for j = 1 to m do begin
            p = read_partition($T_j$ in $p_i$ )
        end
        $kmap_i$ = generate_kmap(p)
        $L^i$ = Apriori($kmap_i$)
   end
// Merge phase
    for( k = 2 ; $L_k^i$ != Ø , i = 1, 2, 3,…….., n ; k++)
    do begin
         $C_k^G$  =  $U_{i=1 \text{ to } n}$ $L_i^k$
    end
// Phase II
    for i = 1 to n do begin
        kmap = kmap + $kmap_i$
    end
    for all candidate c  belongs to $C^G$ compute s(c)
    using kmap.
        $L^G$ = {c belongs to $C^G$ / s(c) >= sigma }

    Answer = $L^G$
}
End
```

IV. Results

The following data set in table 2 has been used to implement the algorithm.

TABLE 2: DATASET

S.No	TID	Itemset
1	T1	I1
2	T2	I1,I3
3	T3	I4
4	T4	I2,I3,I4
5	T5	I2,I3,I4
6	T6	I1
7	T7	I4
8	T8	I3,I4
9	T9	I3,I4
10	T10	I1
11	T11	I4
12	T12	I1,I3
13	T13	I1
14	T14	I2,I3,I4
15	T15	I1,I3

Let,

N = 15
n = 3
m = N/n = 15/3 = 5
Support = 20%

Since each partition consists of 5 transactions so Partition P1 contains transactions from T1 to T5, Partition P2 contains transactions from T6 to T10 and Partition P3 contains transactions from T11 to T15.

Step1 - On the basis of transaction Table2, Karnaugh map matrix for first Partition P1 is obtained (Table 3). The numbers in the cells show the frequency of their allocated item set.

TABLE3: K-MAP1 MATRIX

I1I2 \ I3I4	~I3~I4 C1 00	~I3I4 C2 01	I3~I4 C3 10	I3I4 C4 11
~I1~I2 R1 00	0	1	0	0
~I1I2 R2 01	0	0	0	2
I1~I2 R3 10	1	0	1	0
I1I2 R4 11	0	0	0	0

Step2 – Each partition have 5 transactions so for P1 Minimum Support is 20% of 5.

<u>For K=1:</u>

Step2.1- First we identify the candidate set C1 as follows: C1 = {(I1), (I2), (I3), (I4)}

Step2.2- Here pruning is not possible as subsets of 1-itemsets are not present. So C1 will remain unchanged.

Step2.3- Then we calculate the support count of single items. They all are the elements of set C1.

From (3a), Supp(I1) = (1+0+1+0) + (0+0+0+0) = 2

From (3a), Supp(I2) = (0+0+0+2) + (0+0+0+0) = 2

From (3b), Supp(I3) = (0+0+1+0) + (0+2+0+0) = 3

From (3b), Supp(I4) = (1+0+0+0) + (0+2+0+0) = 3

I1, I2, I3, I4 $\in$ C1

Step2.4- Now compare the support count of single itemsets with *min_sup* to generate frequent single item. For example:

Supp(I1)$\rightarrow$2, Supp(I2)$\rightarrow$2, Supp(I3)$\rightarrow$3, Supp(I4)$\rightarrow$3

Step2.5- Here all the single items have Support > *min_sup*. So all 4 items are frequent. Thus,

L1 = {(I1), (I2), (I3), (I4)}

We can make C2 by all possible combinations of frequent 1-itemsets.

For K=2:

Step2.6- Likewise, Candidate set C2 of 2-itemsets contains following items:
C2 = {(I1I2), (I3I4), (I2I4), (I2I3), (I1I3), (I1I4)}

Step2.7- Again as all the subsets of 2-itemsets are frequent in themselves as shown in step2.5, none of the itemset is pruned from C2. So C2 will remain unchanged.

Step2.8- Calculate the support count for all possible 2-itemsets from C2.
Supp(I1I2) = 0, Supp(I3I4) = 2, Supp(I2I4) = 2, Supp(I2I3) = 2, Supp(I1I3) = 1, Supp(I1I4) = 0

Step2.9- Now we compare the support counts with min_supp to generate frequent 2-itemsets. For example:
Supp(I1I2)$\rightarrow$0, Supp(I3I4)$\rightarrow$2, Supp(I2I4)$\rightarrow$ 2, Supp(I2I3)$\rightarrow$2, Supp(I1I3)$\rightarrow$1 , Supp(I1I4)$\rightarrow$0

Step2.10-(I3I4) (I2I4) (I2I3) (I1I3) are frequent itemsets and (I1I2) (I1I4) are not frequent. So in next step we will make C3 considering only (I3I4) (I2I4) (I2I3) (I1I3) itemsets. Thus,
L2 = {(I3I4), (I2I4), (I2I3), (I1I3)}

For K=3:

Step2.11- Using above frequent 2-itemsets, Candidate set C3 of size 3-itemsets can be identified as follows: C3 = {(I2I3I4)}

Step2.12- As all the subsets of 3-itemsets are frequent in themselves as shown in step2.10, none of the itemset is pruned from C3. So C3 will remain unchanged.

Step2.13- Support count of itemset (I2I3I4) is :

Supp(I2I3I4) = 2

Step2.14 - Now we compare the support counts with min_supp to generate frequent 3-itemsets.

Step2.15- As supp>*min_supp*. So the frequent itemset is (I2I3I4).Thus, L3 = {(I2I3I4)}

For K=4:

Step2.16- Now the candidate set C4 will not contain any element so we can stop the algorithm and frequent itemsets can be obtained as follows:

L^1 = L1 U L2 U L3
= {(I1), (I2), (I3), (I4), (I3I4), (I2I4), (I2I3),
(I1I3), (I2I3I4)}

Step3 – Likewise, for Partition P2, K-Map is shown in Table 4:

TABLE4: K-MAP2 MATRIX

I1I2 \ I3I4	~I3~I4 C1 00	~I3I4 C2 01	I3~I4 C3 10	I3I4 C4 11
~I1~I2 R1 00	0	3	0	2
~I1I2 R2 01	0	0	0	3
I1~I2 R3 10	4	0	3	0
I1I2 R4 11	0	0	0	0

Step3.1 – Similarly, Frequent itemsets for P2 are:
L^2 = {(I1), (I3), (I4), (I3I4)}

Step4 – Likewise, for Partition P3, K-Map is shown in Table 5:

TABLE5: K-MAP3 MATRIX

I1I2 \ I3I4	~I3~I4 C1 00	~I3I4 C2 01	I3~I4 C3 10	I3I4 C4 11
~I1~I2 R1 00	0	1	0	0
~I1I2 R2 01	0	0	0	1
I1~I2 R3 10	1	0	2	0
I1I2 R4 11	0	0	0	0

Step4.1 – Similarly, Frequent itemsets for P3 are:
L^3 = {(I1), (I2), (I3), (I4), (I1I3), (I2I3),
(I2I4), (I3I4), (I2I3I4)}

Step5 – Generate generalized K-Map (Table 6) for all 15 transactions by adding K-Maps of all partitions as shown below:

TABLE6: K-MAP MATRIX

I1I2 \ I3I4	~I3~I4 C1 00	~I3I4 C2 01	I3~I4 C3 10	I3I4 C4 11
~I1~I2 R1 00	0	1	0	2
~I1I2 R2 01	0	0	0	0
I1~I2 R3 10	2	0	0	0
I1I2 R4 11	0	0	0	0

Step6 – Now min_supp for all 15 transactions is 20% of 15.

$C = L^1 \cup L^2 \cup L^3$

= {(I1), (I2), (I3), (I4), (I3I4), (I2I4), (I2I3), (I1I3), (I2I3I4)}

Step6.1 – Calculate support count for all itemsets in C from K-map in table 6.

Step6.2 – Support of all the items is greater than min_supp i.e 3. So, all items of C are frequent. Thus,
L = {(I1), (I2), (I3), (I4), (I3I4), (I2I4), (I2I3), (I1I3), (I2I3I4)}

Step7 – Set of frequent patterns is represented by L obtained in step 6.2.

Step8 – Stop.

Experimental Results:

An interface for the above algorithm was created in MATLAB Version 7.7.0. After applying the algorithm over the transactions discussed above in table 2, along with 3 partitions and 0.2% confidence, following results appeared as shown in table 7.

TABLE7: COMPARISON OF EXECUTION TIME OF PARTITION AND PROPOSED ALGORITHM WITH 15 TRANSACTIONS

Comparison between Partition and K-Partition Algorithm		
Support	Partition algorithm (Time in seconds)	K-Partition algorithm (Time in seconds)
10%	6.776813	0.279434
20%	6.65218	0.27204
30%	6.575818	0.232803
40%	6.474813	0.178912
50%	5.564497	0.175738

Then, we have created a dataset containing 100 transactions containing combinations of 10 different items. After applying same algorithm, with 10 partitions and 0.4% confidence, following results appeared as shown in table 8.

TABLE8: COMPARISON OF EXECUTION TIME OF PARTITION AND PROPOSED ALGORITHM WITH 100 TRANSACTIONS

Comparison between Partition and K-Partition Algorithm		
Support	Partition algorithm (Time in seconds)	K-Partition algorithm (Time in seconds)
10%	22.993544	16.121244
20%	20.5149	13.670069
30%	20.071476	13.591382
40%	8.629537	1.655186
50%	6.297886	0.204197

The tabular result shows clearly that K-Partition algorithm is more efficient than Partition algorithm.

V. CONCLUSION

In this paper, we propose a K-Partition algorithm which is actually advancement in the traditional Partition algorithm. This is achieved with the introduction of Karnaugh Map model. Designing of K-Maps require only one scan of the database, thus, improving the efficiency of processor in terms of CPU elapsed time. Results shows that the approach is not only effective in finding out the frequency of various items but also aim to the prospect item mining with adaptive manner.

VI. FUTURE WORK

This algorithm K-Partition can further be improved by reducing the number of items in candidate set, using various soft computing techniques.

VII. REFERENCES

[1] F. Berzal, J.C. Cubero, N. Marin, J.M. Serrano, " An efficient method for association rule mining in relational databases", Elserier Data & Knowledge, Engineering, pp. 47–64, 2001.

[2] S. Brin, R. Motwani, C. Silverstein, "Beyond market baskets: generalizing association rules to correlations", ACM SIGMOD Conference on Management of Data, Tuscon, AZ, pp. 265–276, May 1997.

[3] Lee S. and Cheung D, " Maintenance of discovered association rules When to update?", In Research Issues on Data Mining and Knowledge Discovery, 1997.

[4] Jurgen M. Jams Fakultat fur Wirtschafts- irnd, "An Enhanced Apriori Algorithm for Mining Multidimensional Association Rules", 25th Int. Conf. Information Technology interfaces ITI Cavtat, Croatia, 2003.

[5] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li.New, "Algorithms for Fast Discovery of Association Rules", In Proc. of the 3rd Int'l Conference on Knowledge Discovery and Data Mining, Newport Beach, Cal-ifornia, Aug. 1997.

[6] Thomas S., Bodagala S., Alsabti K., and Ranka S., "An efficient algorithm for the incremental updating of association rules", In Proc. of the 3rd ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining,1997.

[7] Klemetinen, L., Mannila, H., Ronkainen, P., "Finding interesting rules from large sets of discovered association rules", Third International Conference on Information and Knowledge Management Gaithersburg, USA, pp.401-407, 1994.

[8] Ravindra Patel, D. K. Swami and K. R. Pardarsani, "Lattice Based Algorithm for Incremental Mining of Association Rules", International Journal of Theoretical and Applied Computer Sciences, Volume 1 Number 1 Journal Computer Science, USA , pp. 119–128, 2006.

[9] K. Saruladha, Dr. G. Aghila, B. Sathiya, "A Partitioning Algorithm for Large Scale Ontologies", Volume 1 Number 9, ICRTIT, pp. 526-530, 2012.

[10] Hui Cao, Gangquan Si, Yanbin Zhang, and Lixin Jia, "A Density-based Quantitative Attribute Partition Algorithm for Association Rule Mining on Industrial Database", American control conference, Westin Seattle Hotel, Seattle, Washington, USA, June 11-13, pp. 75-80, 2008.

[11] Neelu Khare, Neeru Adlakha and K. R. Pardasani , "Karnaugh Map Model for Mining Association Rules in Large Databases",International Journal of Computer and Network Security, Volume 1 Number 2 , pp.16-21, 2009.

[12] "wilkipedia," [Online]. Available: www.wilkipedia.com [Accessed: June. 25, 2012].