Optimized Frequent Pattern Mining for Classified Data Sets

A Raghunathan [a] **,** ***K Murugesan*** [b]

[a] *Deputy General Manager-IT, Bharat Heavy Electricals Ltd, Tiruchirappalli, India*
[b] *Assistant Professor of Mathematics, National Institute of Technology, Tiruchirappalli,, India*

Abstract

Mining frequent patterns in data is a useful requirement in several applications to guide future decisions. Association rule mining discovers interesting relationships among a large set of data items. Several association rule mining techniques exist, with the Apriori algorithm being common. Numerous algorithms have been proposed for efficient and fast association rule mining in data bases, but these seem to only look at the data as a set of transactions, each transaction being a collection of items. The performance of the association rule technique mainly depends on the generation of candidate sets. In this paper we present a modified Apriori algorithm for discovering frequent items in data sets that are classified into categories, assuming that a transaction involves maximum one item being picked up from each category. Our specialized algorithm takes less time for processing on classified data sets by optimizing candidate generation. More importantly, the proposed method can be used for a more efficient mining of relational data bases.
Keywords: Data mining, association rule, Apriori algorithm, transactions, frequent items, itemsets.

1. Introduction

Mining frequent patterns or trends in data is an interesting, useful and important requirement in several applications to guide future decisions. Association rule (AR) mining discovers interesting relationships among a large set of data items by finding all frequent itemsets. A very useful application is in Web services and e-Commerce, in areas like web site navigation analysis and online orders and sales transactions analysis, to know customer preferences and trends. There are several algorithms for association rule mining such as Apriori [4] and FP growth [13]. Here the input data is a collection of transactions, each transaction containing one or more items chosen from a warehouse of items, each item occurring once.

Ever since AR mining was proposed for market survey in [1], it has been the subject of intense research, and has spawned several applications. One of the common and useful algorithms is the Apriori algorithm [4], and there have since been numerous algorithms and techniques suggested for its efficient and fast implementation to improve performance. Also, a lot of algorithms have also been proposed for its implementation in databases, as the database is regarded as a collection of transactions. One aspect that we found in the existing algorithms, while applying to databases, esp. relational, is that they treat each row

(transaction) only as just a set of items. However, we feel that performance of the algorithm could be improved if we treat the data items as being classified into categories. In other words, we look upon a relational database as a set of rows, each row (tuple) is a transaction set of selected data items each coming from a different category (attribute), assuming a maximum of one item per category. Thus the database constitutes a classified data set. We have found that, when the existing Apriori algorithm is suitably modified to take into consideration this view of databases, a significant performance of the algorithm results.

Apriori algorithm iteratively searches for frequent itemsets in the given transaction database. At each iteration, a new candidate set of itemsets is generated based on the output of the previous iteration. The performance of Apriori depends on several factors such as size of the input database, transaction size, and the operations of candidate sets generation and selection based on the count of minimum support of the generated candidate sets in the transaction database. The generation of candidate sets itself consists of two operations, viz., joining and pruning. The joining operation produces the candidate sets for the current iteration and the pruning operation reduces the size of the candidate sets based on prior knowledge. Both these operations are computational expensive. Techniques proposed to improve Apriori have concentrated on both algorithmic and implementation aspects. Most concentration has been on the phases of pruning, counting, partitioning the input data set, data structures used, storage and access, and reducing the number of passes over the input database. In this paper we present a modified Apriori algorithm for classified data sets that optimizes candidate generation by reducing the number of iterations in the joining phase, and also the number of candidate itemsets generated. This specialised algorithm not only takes less time than the general version, but also yields a more efficient method of mining relational databases.

2. Earlier work

The problem of mining association rules was introduced by Agrawal, et al. in [1], who also brought out the Apriori algorithm [4] for market-based data. Subsequently, there have been many efficient and fast AR algorithms proposed, including improved Apriori implementations. Partitioning [18] was used to speed up Apriori in respect of I/O, while sampling [20] was adopted to reduce processing. Both these algorithms sought to achieve reduced number of passes over the input database. An algorithm to improve pruning and counting in Apriori was suggested in [16]. A Hash-based algorithm for candidate set generation is suggested in [17]. Other fast and efficient methods were proposed in [3], [6], [7], [15], [19]. Techniques for mining large databases with performance are discussed in [2],

[5], [8], [9], [11], [15]. Useful techniques for mining frequent sets are discussed in [10] and [14].

3. Frequent items mining using Apriori algorithm

There are several algorithms to find frequent patterns in data sets. One of the common and useful algorithms is the Apriori algorithm, whose pseudocode is given in Fig. 1 adapted from [12]. It uses prior knowledge of frequent itemset properties and employs an iterative approach using a level-wise search based on candidate generation. This technique is applied to a general data set of items. The input is a database of transactions containing items drawn from a set of items. Each transaction contains an itemset. The output of the algorithm is a set of itemsets that frequently occur in the input transactions [12].

There are several applications involving special cases of data sets that may include data items classified into categories, such as in a departmental store. Here a transaction involves picking up items from each category. Existing AR mining algorithms treat each transaction as a set of items, without regard to the classification. In this paper, we propose a modification of the Apriori algorithm to apply to such classified data sets, leading to faster results. We assume that at most one item is picked from each category. As explained earlier, this assumption is useful to apply our algorithm to relational databases.

Let us illustrate our case with an example. Let us consider a small computer shop selling classified items as in Table 1. Table 2 shows a selection of these items under various categories that participate in a set of transactions for illustration. The items have been assigned unique IDs for use in the algorithms. Table 3 shows a sample database containing transactions involving these items.

Table 1 – Sample items on sale in a computer shop

Category	Items
Computer	Desktop, Laptop, Tablet
Peripherals	Printer, Scanner, All-in-one
Software	Antivirus, Games, Utility, Educational
Accessories	Media, UPS, Flash Drive, Modem, Speaker

We shall apply the standard Apriori algorithm given in Fig. 1 to this transactional data D. Since the algorithm is a general one that treats all items in one pool, the transactional items are considered irrespective of their category.

Fig. 1. The Apriori Algorithm for mining and discovering frequent itemsets from data.

Algorithm: Apriori. Find frequent itemsets using an iterative level-wise approach based on candidate generation.

Input: Database *D* of transactions; minimum support threshold, *min_sup*.

Output: *L*, frequent itemsets in *D*.

Method:

```
1.  L1= find_frequent_1-itemsets(D, min_sup);
2.  for (k = 2;Lk-1 ≠ Ø; k++)
3.  {
4.      Ck = apriori_gen(Lk-1);
5.      for each transaction t ∈ D // scan D for counts
6.      {
7.          Ct = subset(Ck, t);
8.          for each candidate c ∈ Ct
9.          c.count++;
10.     }
11.     Lk = {c ∈ Ck| c.count ≥ min_sup);
12. }
13. return L = ∪ kLk;

procedure apriori_gen (Lk-1)
1.  for each itemset l1 ∈ Lk-1
2.      for each itemset l2 ∈ Lk-1
3.          if (l1[1] = l2[1]) ^ (l1[2] = l2[2]) ^ …^ (l1[k-2] = l2[k-2]) ^ (l1[k-1] < l2[k-1])
4.          then
5.          {
6.              c = l1 ⋈ l2; // join step: generate candidates
7.              if has_infrequent_subset(c, Lk-1) then
8.                  delete c; // prune step: remove infrequent candidates
9.              else add c to Ck;
10.         }
11. return Ck;

procedure has_infrequent_subset (c:candidate k-itemset; Lk -1:frequent (k - 1) itemsets)
1.  for each (k - 1)-subset s of c
2.      if s ∉ Lk-1 then return TRUE;
3.  return FALSE;

procedure find_frequent_1-itemsets(D, min_sup)
//      Form C1, the candidate 1-itemset list from D, with the items uniquely listed in sorted order.
1.  C1 = {c ∈ D};
2.  for each transaction t ∈ D // scan D for counts
3.  {
4.      Ct = subset(C1, t);
5.      for each candidate c ∈ Ct
6.          c.count++;
7.  }
8.  L1 = {c ∈ C1| c.count >= min_sup);
return L1;
```

Table 2 – Items with unique IDs

Category	Items	ItemID
Computer	Desktop	I1
	Laptop	I2
Peripherals	Printer	I3
Software	Games	I4
Accessories	UPS	I5
	Flash Drive	I6

Table 3 – Transactional data for a Computer shop

TID	List if Item IDs
T1	I1, I3, I5
T2	I3, I5
T3	I2, I3, I6
T4	I1, I4, I6
T5	I2, I6
T6	I2, I3, I4, I6
T7	I2, I4
T8	I1, I3, I4, I5
T9	I3, I6

Applying the normal Apriori Algorithm given in Fig. 1 to the transactions in Table 3, we generate the candidate itemsets and frequent itemsets, where the minimum support count is assumed to be 2, i.e., 2/9 = approx. 20%. It is assumed that items within a transaction or itemset are sorted in lexicographic order. The outputs are shown in Tables 4 – Table 12.

Table 4

C_1

Itemset	Sup. count
{I1}	3
{I2}	4
{I3}	6
{I4}	4
{I5}	3
{I6}	5

Table 5

L_1

Itemset	Sup. count
{I1}	3
{I2}	4
{I3}	6
{I4}	4
{I5}	3
{I6}	5

Table 6

C_2

Itemset
{I1,I2}
{I1,I3}
{I1,I4}
{I1,I5}
{I1,I6}
{I2,I3}
{I2,I4}
{I2,I5}
{I2,I6}
{I3,I4}
{I3,I5}
{I3,I6}
{I4,I5}
{I4,I6}
{I5,I6}

Table 7

C_2

Itemset	Sup. count
{I1,I2}	0
{I1,I3}	2
{I1,I4}	2
{I1,I5}	2
{I1,I6}	1
{I2,I3}	2
{I2,I4}	2
{I2,I5}	0
{I2,I6}	3
{I3,I4}	2
{I3,I5}	3
{I3,I6}	3
{I4,I5}	1
{I4,I6}	2
{I5,I6}	0

Table 8

L_2

Itemset	Sup. count
{I1,I3}	2
{I1,I4}	2
{I1,I5}	2
{I2,I3}	2
{I2,I4}	2
{I2,I6}	3
{I3,I4}	2
{I3,I5}	3
{I3,I6}	3
{I4,I6}	2

Table 9

C_3

Itemset
{I1,I3,I4}
{I1,I3,I5}
{I1,I4,I5}
{I2,I3,I4}
{I2,I3,I6}
{I2,I4,I6}
{I3,I4,I5}
{I3,I4,I6}
{I3,I5,I6}

Table 10

C_3

Itemset
{I1,I3,I4}
{I1,I3,I5}
{I2,I3,I4}
{I2,I3,I6}
{I2,I4,I6}
{I3,I4,I6}

Table 11

C_3

Itemset	Sup.count
{I1,I3,I4}	1
{I1,I3,I5}	2
{I2,I3,I4}	1
{I2,I3,I6}	2
{I2,I4,I6}	1
{I3,I4,I6}	1

Table 12

L_3

Itemset	Sup.count
{I1,I3,I5}	2
{I2,I3,I6}	2

At the end, the candidate set of 4-itemsets, $C_4 = \emptyset$, and the algorithm terminates. All the frequent itemsets in the transaction database, L_1, L_2 and L_3, have been found as shown in Tables 5, 8 and 12.

4. **Working of the algorithm**

The Apriori algorithm works as follows: Step 1 (Fig. 1) invokes the find_frequent_1_itemset procedure, which scans the input transaction data D and forms C_1, the candidate 1-itemsets, which is nothing but the list of unique individual items in D. It then computes the number of occurrences of each item and produces L_1, which contains the items occurring with the minimum support count given, *min_sup* (assumed to be 2 in this example). Successive candidate itemsets C_k are then generated iteratively (steps 2-10) by *joining* $\boldsymbol{L_{k-1}}$ with $\boldsymbol{L_{k-1}}$ (step 6). The apriori_gen procedure generates the candidate sets and eliminates those having a subset that is not frequent. This is called *pruning* and is done by invoking the has_infrequent_subset procedure. Finally, all those candidates satisfying minimum support form the set of frequent itemsets, L_1 through L_3. Thus we can see that the itemsets {I1, I3, I5} and {I2, I3, I6} occur 2 times each in the transaction dataset D.

5. **Observations with existing algorithm**

We have used the Apriori algorithm shown in Fig. 1 to mine the input data set and got the result. In the candidate 2-itemset set C_2 (Table 6) generated by the algorithm from L_1 (Table 5), the presence of itemsets {I1, I2} and {I5,I6} may be noted. Each of these itemsets has items drawn from the same category, i.e., in itemset {I1, I2}, both items I1 and I2 belong to the category Computer and similarly both the items I5 and I6 belong to the category Accessories. Since we have assumed that not more than one item is chosen from a category, we would not like these itemsets to be generated as candidates. They were generated because the apriori_gen procedure considers all itemsets of L_{k-1} for *k*-itemset candidate generation, regardless of categories, and always executes *n* x *n* times, where *n* is the size of L_{k-1}. Hence we bring out a modified algorithm taking into consideration special cases when the itemsets contain classified items under several categories, with a transaction involving no more than one item from each category. This method could not only be applied to special applications, but also would reduce the no. of iterations in generating the candidate itemsets.

6. **Modified algorithm for Classified Data Sets**

We now present a modified version of the Apriori algorithm in Fig. 2 to mine classified data sets, with items falling under distinct categories, assuming that a transaction involves maximum one item in each category.

Fig. 2. Modified Apriori algorithm for mining classified data sets.

Algorithm: Apriori_Class. Find frequent itemsets belonging to different categories using an iterative level-wise approach based on candidate generation.

Input: Database *D* of transactions involving classified items, min. support threshold, *min_sup*.

Output: *L*, frequent itemsets in *D*.

// This algorithm considers items classified under various identified categories,
// eg., A,B,C, etc. The items within each category bear unique IDs, eg., A1,A2,D3,E4, etc.
// A typical transaction comprises zero or max. one item selected from each category.
// Hence, the transaction database D is a matrix, with rows representing transactions and
// columns containing item IDs under various categories.
// Items within a transaction or itemset are assumed to be sorted in lexicographic order.

Method:

```
1.  L1= find_frequent_1-itemsets(D, min_sup);
2.  for (k = 2;Lk-1 <> Ø; k++)
3.  {
4.        Ck = apriori_gen(Lk-1);
5.        for each candidate  c ∈ Ck
6.             for each transaction t ∈ D
7.                  if  c∈ t then c.count++;
8.        Lk = {c ∈ Ck| c.count >= min_sup);
9.  }
10. return L = ∪ kLk;

procedure find_frequent_1-itemsets(D, min_sup)
//        Form C1, the candidate 1-itemset list from D, with the items uniquely listed in sorted order.
9.  C1 = {c ∈ D};
10. for each candidate  c ∈ C1
11.       for each transaction t ∈ D
12.            if c ∈ t then c.count++;
13. L1 = {c ∈ C1| c.count >= min_sup);
return L1;

procedure apriori_gen(Lk-1)
1.  if k = 2 then
2.  begin
3.     ub = count(Lk-1);
4.     for (l1 = 1; l1 < ub; l1++)
5.     begin
6.        l2 = l1 + 1;
7.        repeat until (category(l2) > category(l1)) or (l2 > ub)
8.              l2++;
9.        if (l2 <= ub) and (category(l2[1]) > category(l1[1])) then
10.       begin
11.            lb = l2;
12.            for (j = lb; j ≤ ub; j++)
13.            {
14.               l2 = Lk-1[lb];
15.               c = l1 ⋈ l2;
16.               if has_infrequent_subset(c, Lk-1) then delete c;
17.               else add c to Ck;
18.            }
19.       end;
20.    end;
21. end;
22. else if k > 2 then
23.     begin
```

```
24.          ub = count(L_{k-1});
25.          for (l_1 = 1; l_1 < ub; l_1++)
26.          begin
27.              l_2 = l_1 + 1;
28.              while  (l_1[1] = l_2[1]) ^ (l_1[2] = l_2[2]) ^ …^ (l_1[k-2] = l_2[k-2]) ^ (l_1[k-1] < l_2[k-1]) do
29.                 begin
30.                     if (category(l_2[k-1]) > category(l_1[k-1])) then
31.                         begin
32.                             c = l_1 ⋈ l_2;
33.                             if has_infrequent_subset(c, L_{k-1}) then delete c;
34.                             else add c to C_k;
35.                         end;
36.                         l_2 ++;
37.                 end;
38.          end;
39.      end;
40.  return C_k;

procedure has_infrequent_subset(c:candidate k-itemset; L_{k-1}:frequent (k-1) itemsets)
1.   for each (k-1)-subset s of c
2.        if s ∉ L_{k-1} then return TRUE;
3.   return FALSE;
```

We now apply the modified algorithm to our example. Table 13 shows the same items shown earlier in Table 1, but now assigned distinct IDs under four categories A - D. Table 14 shows the transactional data of Table 2 as a selection from the various categories.

Table 13 – Items in Computer shop with unique IDs

A - COMPUTER		B - PERIPHERALS		C – SOFTWARE		D - ACCESSORIES	
A1	Desktop	B1	Printer	C1	Games	D1	UPS
A2	Laptop					D2	Flash Drive

Table 14 – Transactional data for Computer shop

TID	A - COMPUTER	B - PERIPHERALS	C - SOFTWARE	D - ACCESSORIES
T1	A1	B1		D1
T2		B1		D1
T3	A2	B1		D2
T4	A1		C1	D2
T5	A2			D2
T6	A2	B1	C1	D2
T7	A2		C1	
T8	A1	B1	C1	D1
T9		B1		D2

Applying the modified Apriori Algorithm given in Fig. 2 to Table 14, we generate the candidate itemsets and frequent itemsets, with minimum support count 2. The outputs are shown in Tables 15 through 23.

Table 15

C_1

Itemset	Sup. count
{A1}	3
{A2}	4
{B1}	6
{C1}	4
{D1}	3
{D2}	5

Table 16

L_1

Itemset	Sup. count
{A1}	3
{A2}	4
{B1}	6
{C1}	4
{D1}	3
{D2}	5

Table 17

C_2

Itemset
{A1,B1}
{A1,C1}
{A1,D1}
{A1,D2}
{A2,B1}
{A2,C1}
{A2,D1}
{A2,D2}
{B1,C1}
{B1,D1}
{B1,D2}
{C1,D1}
{C1,D2}

Table 18

C_2

Itemset	Sup.count
{A1,B1}	2
{A1,C1}	2
{A1,D1}	2
{A1,D2}	1
{A2,B1}	2
{A2,C1}	2
{A2,D1}	0
{A2,D2}	3
{B1,C1}	2
{B1,D1}	3
{B1,D2}	3
{C1,D1}	1
{C1,D2}	2

Table 19

L_2

Itemset	Sup. count
{A1,B1}	2
{A1,C1}	2
{A1,D1}	2
{A2,B1}	2
{A2,C1}	2
{A2,D2}	3
{B1,C1}	2
{B1,D1}	3
{B1,D2}	3
{C1,D2}	2

Table 20

C_3

Itemset
{A1,B1,C1}
{A1,B1,D1}
{A1,C1,D1}
{A2,B1,C1}
{A2,B1,D2}
{A2,C1,D2}
{B1,C1,D1}
{B1,C1,D2}

Table 21

C_3

Itemset
{A1,B1,C1}
{A1,B1,D1}
{A2,B1,C1}
{A2,B1,D2}
{A2,C1,D2}
{B1,C1,D2}

Table 22

C_3

Itemset	Sup.count
{A1,B1,C1}	1
{A1,B1,D1}	2
{A2,B1,C1}	1
{A2,B1,D2}	2
{A2,C1,D2}	1
{B1,C1,D2}	1

Table 23

L_3

Itemset	Sup.count
{A1,B1,D1}	2
{A2,B1,D2}	2

As before $C_4 = \emptyset$, and the algorithm terminates when all the frequent itemsets in the transaction database have been found.

7. Working of the modified algorithm

The modified Apriori algorithm in Fig. 2 uses a revised apriori_gen procedure in which C_k generation is considered separately for cases k=2 and k>2. While generating C_2 from L_1 (Fig. 2 procedure apriori_gen steps 7-19), two items of L_1 are joined only when their categories are different. Thus the itemsets {A1, A2} and {D1, D2} are not generated in the C_2 given in Table 17. Moreover, as the items in L_1 are in sorted lexicographic order, we restrict the scope of items in L_1 for comparison in each iteration, reducing the range. Likewise, in generating C_3 from L_2 through steps 25-34, two items are joined only when their categories are the same. Thus we eliminate the generation of itemset {B1, D1, D2} in Table 20, and this is done by step 30 in Fig. 2. Again the range of items to be compared in each iteration is restricted in step 28. Steps 6 and 27 also reduce the number of items in L_{k-1} for comparison by 1, thus reducing the number of iterations still further.

8. Comparison of the Algorithms

We now compare the two algorithms and the results obtained for our example discussed above for the same transaction data set.

Eliminating itemsets

The original algorithm-generated C_2 given in Table 6 is compared with the C_2 in Table 17 generated by the modified algorithm of Fig. 2. The former generates itemsets {I1,I2} and {I5,I6} which contain items that are part of the same category, as observed above. However, the corresponding itemsets viz., {A1,A2} and {D1,D2} are not generated in the C_2 given in Table 17, as explained above. Similarly, while generating C_3, the itemset {I3,I5,I6} output in Table 9 is eliminated in the new algorithm by filtering out the itemset {B1,D1,D2} in Table 20.

Optimising iterations

As explained in the previous section, when compared to the original algorithm, the new algorithm has far fewer iterations, as implemented through the modified apriori_gen procedure in Fig.2. In the original procedure in Fig. 1, steps 1 and 2 iterate for each item of L_{k-1} with itself, thus executing n2 times, where n is the size of L_{k-1}. Whereas in the modified procedure in Fig. 2, not only the outer loop executes one less time than before (steps 4 and 25), but also fewer itemsets are compared in each iteration in the inner loop. Hence the candidate sets generation is done much faster.

Table 24 summarises the results of the comparison for generation of each candidate set C_k. It can be noted that the original algorithm takes n^2 iterations whereas our modified algorithm takes a figure of m iterations where m is O(n *log* n). That is, the modified algorithm takes O(n *log* n) time compared to O(n^2), which is very significant. Also, the number of generated intermediate itemsets in successive iterations (during join operation before pruning) is also reduced, as seen from the last two columns.

Table 24 – Performance comparison of the two Apriori algorithms for the sample transaction data

Candidate set *Ck*	Set size |*Lk-1*|	No. of iterations (original)	No. of iterations (modified)		No. of itemsets generated (original)	No. of itemsets generated (modified)
	n	***n*²**	***m***	***n log n***		
C2	6	36	15	10.8	15	13
C3	10	100	18	23.0	9	8
C4	2	4	1	1.4	0	0
Total	**18**	**140**	**34**	**35.2**	**24**	**21**

9. Performance Analysis

We now analyse the comparative performance of the two algorithms for various cases, based on our testing, involving transactions of various sizes of elements and categories. Table 25 shows the results. Here we note that the total no. of iterations increases with an increase in either the no. Elements or the no. of categories. In each case the modified algorithm generates fewer itemsets in fewer iterations compared to the original algorithm. We plot the above results in Charts 1 and 2, which clearly show the improved performance.

Table 25 – Performance comparison of the two Apriori algorithms for various transaction data sizes

Case No.	No. Of categories	No. Of elements	Total set size $\|Lk\text{-}1\|$	Total no. of iterations (original)	Total no. of iterations (modified)		Total no. of itemsets generated (original)	Total no. of itemsets generated (modified)
			$\sum n$	$\sum n^2$	$\sum m$	$\sum n \log n$		
1	4	6	18	140	34	35.2	24	21
2	5	11	19	149	47	37.8	38	28
3	3	12	17	157	62	37.2	57	41
4	4	12	23	225	71	50.8	59	49
5	5	12	28	326	78	69.7	62	54

Further, taking a closer look at Table 25, we get some interesting results with regard to transactions involving classified data sets with the same number of elements but distributed under different number of categories. Table 26 shows extracts of Table 25 with only the last three rows – transactions with of elements 12, but different numbers of categories – 3, 4 and 5. We observe that as the no. of categories increases, the no. Of iterations in the modified algorithm gets closer to *n log n*, as given by the column $\sum m - \sum n \log n$. Further, as the column reveals, the new no. of iterations gets much less than the original no. of iterations as the no. of categories. This means that the performance of our modified algorithm increases when the same no. of items is distributed under more categories, i.e., classification of data sets increases. This is a significant result for our modified algorithm. Chart 3 depicts the results based on Table 26.

Table 26 – Performance comparison for data sets with same no. of elements but under different categories

Case No.	No. of elements	No. of categories	Total set size $\|Lk\text{-}1\|$	Total no. of iterations (original)	Total no. of iterations (modified)		Closeness of no. of iterations (modified) to n log n	Diff. in no. of iterations (original - modified)
			$\sum n$	$\sum n^2$	$\sum m$	$\sum n \log n$	$\sum m - \sum n \log n$	$\sum n^2 - \sum m$
3	12	3	17	157	62	37.2	24.8	95
4	12	4	23	225	71	50.8	20.2	154
5	12	5	28	326	78	69.7	8.3	248

Chart 1

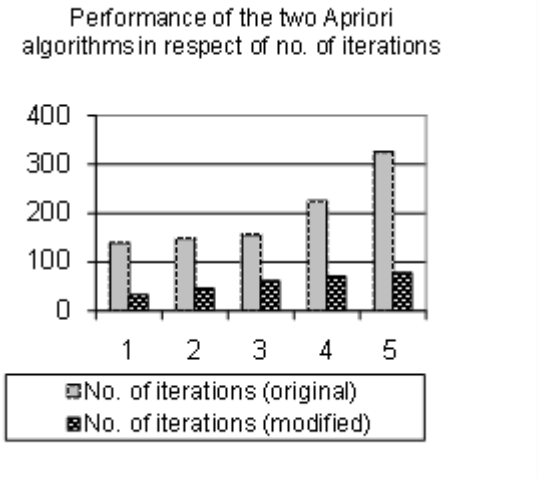


Chart 2

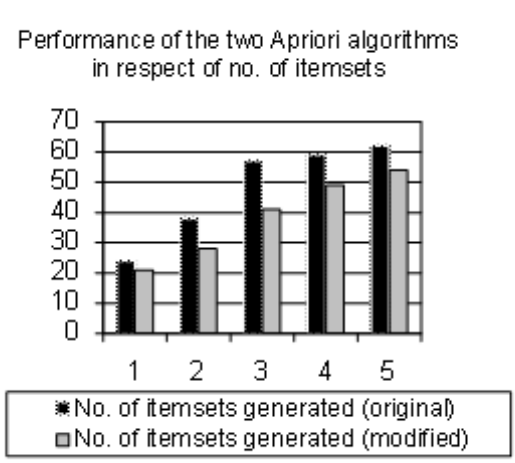
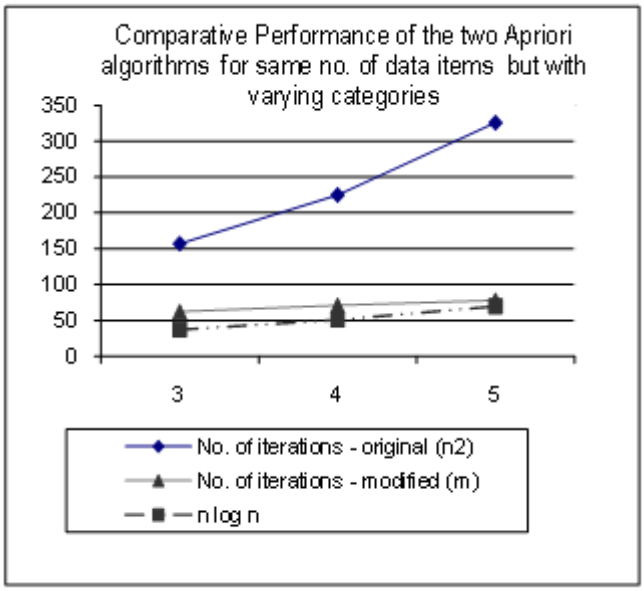


Chart 3



10. Applications to databases

One of the important objectives of our adapting the Apriori association rule mining technique to classified data sets is to apply the concept to the realm of databases to mine frequent occurrences of data items. Let us consider a relational data table for example. A table consists of tuples or rows where each row gives information about a distinct object, each object having values under distinct columnar attributes. Such a table of data values could be considered as a classified data set in our context. Each row can be taken to be a transaction, each attribute a category, and each value to be a data item or an element. Thus, a table is looked upon as a set of transactions involving classified data items under various

categories. This correspondence gives us an easy way to apply our Apriori algorithm to mine frequent occurrences of values in relational databases.

For example, let us look at a sample library catalog of books represented by Table 27.

Table 27 – Sample Library Books Catalog for Finding Frequent Itemsets

Book No.	*Title*	*Author*	*Publisher*	*Year Published*
1	Java	Schildt	TMH	1998
2	C	Schildt	TMH	1997
3	HTML	Collins	Pearson	2000
4	XML	Collins	Pearson	2002
5	C	Balagurusamy	TMH	2001
6	XML	Williamson	TMH	2003
7	C++	Schildt	TMH	2001

Applying our algorithm to the above table, we get the frequent itemsets {Schildt, TMH} and {Collins, Pearson}, assuming *min_sup* to be 2.

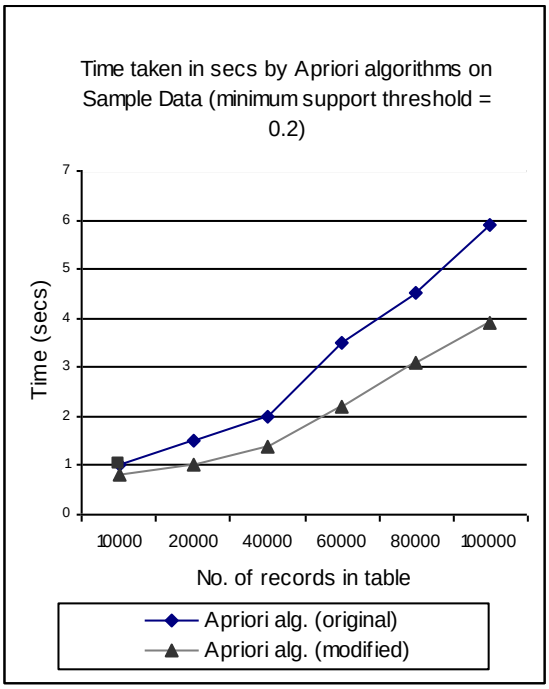


Chart 4

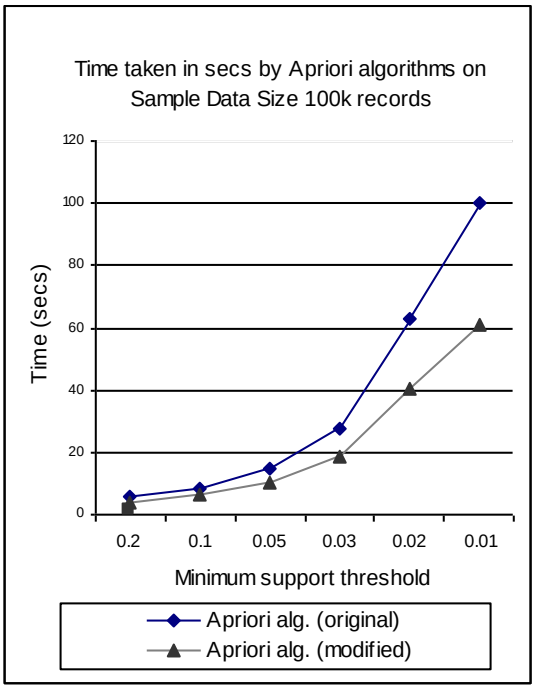


Chart 5

We tested the original and our modified algorithms on sets of relational tables of various sizes and obtained a good performance. The results are plotted in Charts 4 and 5. In Chart 4, the time values are shown for minimum support threshold value of 0.2 for various data volumes (no. of records or transactions). Chart 5 depicts the times taken for a data volume of 100k records for various minimum support threshold values. We observe that there is a significant improvement in the overall execution time by our optimisation.

11. Conclusion and Future Work

In this paper we presented a modified Apriori association rule mining algorithm for discovering frequent patterns in data sets that are classified into categories, assuming a maximum of one item per category. This specialised algorithm takes less time (O(*n log n*)) than the general version (O(n^2)) by reducing the number of iterations as well as the number of candidate sets generated. We also showed that, for the same number of data items, the optimised algorithm performs better when the items are distributed under more categories. While we have taken up the Apriori algorithm for adaptation to classified data sets, our approach could be used in other similar algorithms as well. The proposed algorithm is appropriate to databases to mine frequent occurrences of item values. We are working on extending this approach to mine semi-structured data like XML used in Web services. Future work would focus on applying Association Rule mining techniques to enhance Web services and Web mining with better data management in the areas of storage and search facilities where the techniques have a lot of potential.

References

1. R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In Proc. of ACM SIGMOD COMD, 1993.
2. R. Agrawal, T. Imielinski, and A. Swami. Database Mining: a performance perspective, IEEE TKDE, Dec. 1993.
3. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen and A.I. Verkamo. Fast Discovery of Association Rules. In U.M. Fayyad, et al. Advances in Knowledge Discovery and Data Mining, AAAI/MIT Press, 1996.
4. R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In Proc. Of the 20th VLDB Conf., 1994.
5. R.J. Bayardo Jr. Efficiently mining Long patterns from databases. In Proc. Of the ACM SIGMOD ICMD, 1998.
6. F. Bodon. A Fast Apriori Implementation. In Proc. 1st FIMI 2003.
7. S. Brin, R. Motwani, J.D. Ullman, and T. Tsur. Dynamic itemset counting and implication rules for market based data. ACM SIGMOD Record, 1997.
8. M.S. Chen, J. Han and P.S. Yu. Data Mining: An overview from a database perspective. IEEE Transactions on Knowledge and Data Engineering, 1996.
9. B. Dunke and N. Soparkar. Data organization and access for efficient data mining. In Proc. Of 15th ICDE, 1999.
10. U.M. Fayyad, G. Piatesky-Shapiro, P. Smyth and R. Uthurusamy, editors. Advances in Knowledge Discovery and Data Mining. AAAI Press, 1998.
11. V. Ganti, J. Gehrke, and R. Ramakrishnan. Mining very large databases. IEEE Computer, 1999.
12. J. Han and M. Kamber, Data Mining Concepts and Techniques, Morgan Kaufmann Publishers, 2001.

13. J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. ACM SIGMOD ICMD, 2000.
14. H. Mannila, H. Toivonen and A.I. Verkamo. Efficient algorithms for discovering Association Rules. AAAI Workshop on Knowledge Discovery in Databases, 1994.
15. M.H. Margahny and A.A. Mitwaly. Fast Algorithm for Mining Association Rules. AIML 05 Conf, Egypt.
16. S. Orlando, P. Palmerini and R. Perego. Enhancing the Apriori Algorithm for Frequent Set Counting. DaWak 2001.
17. J.S. Park, M.-S. Chen and P.S. Yu. An effective hash-based algorithm for mining association rules. In Proc. Of ACM SIGMOD ICMD, 1995
18. A. Savasere, E. Omiecinski and S.B. Navathe. An Efficient Algorithm for Mining Association Rules in Large Databases. In Proc. Of 21st VLDB Conf., 1995
19. P. Shenoy, J. Haritsa, S. Sudarshan, G. Bhalotia, M. Bawa, and D. Shah. Turbocharging vertical mining of large databases. In Proc. Of the ACM SIGMOD ICMD, 2000.
20. H. Toivonen. Sampling Large Databases for Association Rules. In The VLDB Journal, 1996.