A Novel Algorithm For Frequent Item-sets Mining Using Reduced Database Scan Approach

Mahesh H. Panchal [1] *Bhagirath P. Prajapati* [2]

Abstract

Association rule mining is one of the techniques of data mining by which valuable but hidden patterns (knowledge) are discovered from large amount of data. Mining of frequent item sets from which association rules are made, is a most challenging task. No. of algorithms had been developed for frequent item set mining, all differ in various aspects. In this paper a novel algorithm which we are calling as frequent 2-base is presented to mine frequent item sets in two database scans. The time taken by frequent 2-base is also compared with respect to various dimensions. Three dimensions are used to compare the time : database size, no. of items and average transaction size.

Keywords : Data Mining, Hidden Pattern, Minimum Support, Frequent Item Sets.

1. Introduction

In competitive market, any organization wants some interesting and hidden conclusions (results), trends, patterns from data stored in database. Data Mining is growing field which provides various methods by which those trends or patterns can be discovered. There are several methods available of data mining including association rule mining, classification, clustering etc. One can choose any one of them or any combination according to the requirement. Association Rule Mining (ARM) is one of the methods which generate closed associations among data items of database. If ARM is applied on market basket data, discovered associations among data items indicates that during one visit of market if customer purchases some items along with them which other items he may purchase.

Mining association rules is a two step process[1][2]. (1) Generating all items which are frequently purchased by customers in market basket problem. The item sets which occur more than user defined threshold value called min_sup are called frequent item sets. (2) Generating all strong association among frequent item sets. The association is called strong if association satisfies another user defined threshold value called min_conf. The problem of finding frequent item sets attracted the attention of many researchers. As a result, at present there exist several algorithms for the same including the ones based on variations of apriori, based on depth first search approach, tree based algorithms[3][4][5].

When any algorithm is finding the frequent item sets, the database remains into secondary storage. Of course, the database is very large so, one of the parameter one should take into consideration in any algorithm is how many i/o access are performed i.e. how many times the database is scanned[3][4]. The apriori algorithm scans the whole database $k+1$ times, where k is the length of longest frequent item set. Some other algorithms are able to find same set of frequent patterns in two scans of database only irrespective of database size and length of frequent item sets.

[1] Asst. Prof., Dept. of Comp Engg, Kalol Institute of Tech & Research Centre, Gujarat. Email : mkhpanchal@gmail.com

[2] Lecturer, Dept. of Comp. Engg., A.D. Patel Institute of Technology, Gujarat. Email : bhagirath_123@yahoo.co.in

In this paper an algorithm called frequent 2-base is shown which requires two scans of database and finds set of frequent patterns in less amount of time compared to other algorithms which also requires same no. of scans. In frequent 2-base first all frequent 2-item sets are derived by scanning the whole database once. From frequent 2-item sets all possible higher order candidate item sets are found. Then from set of candidate item sets of all orders, frequent item sets are taken out by scanning the database second time. It had been verified that no frequent item set of any order is missed out by the algorithm frequent 2-base.

The time taken by frequent 2-base algorithm is compared by various parameters namely size of database (D), no. of data items (N) and average length of transaction (AT). All the statistical results are presented in tabular and graphical form.

2. Association Rule

Discovering association rules is at the heart of data mining. It detects hidden linkages of otherwise seemingly unrelated data. These linkages are rules. Those that exceed a certain threshold are deemed interesting. Interesting rules allow actions to be taken based upon data pattern. They can also help making and justifying decisions. In association rule mining there are two measurements, *support* and *confidence*. The *support* corresponds to the frequency of the pattern while *confidence* indicates rule's strength. A typical example of an association rule created by data mining often termed to as "market basket data" is[6][7]. 80% of customers who purchase bread also purchase butter." An association rule is defined as[16][17].

Let D is set of all distinct data items available in database. T is set of all transactions $t_1, t_2, t_3 \ldots\ldots t_n$. Every transactions ti , where i may be $1,2,3,\ldots.n$ contains some data items from D.

The association rule is of form $X \rightarrow Y$ where X and Y is subset of D and $X \cap Y = \phi$. It is said that association rule $X \rightarrow Y$ holds in database if and only if support of $X \rightarrow Y$ is above min_sup and confidence is above min_conf.

Support of $X \rightarrow Y$ is percentage of transactions out of total transactions, which contain all the data items available in item-sets X and Y. Confidence of $X \rightarrow Y$ is percentage of transactions which contain all the items in Y out of the transactions which contain all the items in X. Suppose bread→milk holds in some market basket database with total 100 transactions with support 60% and confidence 75%. It is interpreted that 60% of transactions out of 100 contain both bread and milk and if bread is purchased than in 75% of those cases milk is also purchased.

Given a user defined minimum support and minimum confidence, the problem of mining association rules is to find all the associations rules whose support and confidence are larger than the minimum support (min_sup) and minimum confidence (min_conf). Thus, the approach can be broken into two sub-problems as follows:

(1) Finding the frequent item sets which have support above the predetermined minimum support.

(2) Deriving strong association rules, based on each frequent item set, which have confidence more than the minimum confidence.

2.1 Frequent Item-Set Mining

The task of discovering all frequent item sets is quite challenging. The search space is exponential in the number of items occurring in the database[3]. The support threshold limits the output to a hopefully reasonable subspace. Also, such databases could be massive, containing millions of transactions, making support counting a tough problem.

Apriori is basic algorithm for frequent item sets mining. In first step it finds L_1 i.e. set of frequent 1-item sets by scanning the database first time. From L_1 it finds set of candidate 2-item sets C_2. Again by scanning the database it finds L_2 from C_2. This process is repeated until no more C_k is possible from L_{k-1} or no further L_k is found from C_k. Apriori uses the prior knowledge by pruning out to generate those candidate item sets whose at least one subset is not frequent. The main drawback of this algorithm is, it is scanning the database k+1 times where k is the length of longest frequent item set. As a result of it i/o cost is increased as the length of longest frequent item set is increased. The most outstanding improvement over Apriori would be a method called FP-growth (frequent pattern growth) that succeeded in eliminating candidate generation[8][9]. It adopts a divide and conquer strategy by (1) compressing the database representing frequent items into a structure called FP-tree (frequent pattern tree) that retains all the essential information and (2) dividing the compressed database into a set of conditional databases, each associated with one frequent item set and mining each one separately. It scans the database only twice. In the first scan, all the frequent items and their support counts (frequencies) are derived and they are sorted in the order of descending support count in each transaction. In the second scan, items in each transaction are merged into a prefix tree and items (nodes) that appear in common in different transactions are counted. Each node is associated with an item and its count. Nodes with the same label are linked by a pointer called node-link. Since items are sorted in the descending order of frequency, nodes closer to the root of the prefix tree are shared by more transactions, thus resulting in a very compact representation that stores all the necessary information. Pattern growth algorithm works on FP-tree by choosing an item in the order of increasing frequency and extracting frequent item sets that contain the chosen item by recursively calling itself on the conditional FPtree. FP-growth is an order of magnitude faster than the original Apriori algorithm.

3. Frequent 2-base Algorithm

This section represents the algorithm frequent 2- base. The complete algorithm is decomposed into following phases: (1) generation of candidate 2-item sets. (2) finding frequent 2-item sets. (3) generating all possible higher order candidate item sets. (4) generation of all higher order frequent item sets. Out of these four phases, first and third require database scan. The output generated by each phase is given as input to next phase. Input of first phase is original database and output of last phase is frequent item sets.

3.1 Generation Of Candidate 2-item Sets

Section 3.1.1 shows pseudo code to generate candidate 2-item sets from given market basket database. That code is explained in section 3.1.2.

3.1.1 Pseudo Code

```
1. temp = null.
2. Repeat for i=1 to n
3. Repeat for each candidate 2-item set X2
4.      If (X2 is not in temp)
5.              X2.count = 1
6.              temp = temp U X2
7.      If (X2 is in temp)
8.              X2.count = X2.count + 1
9. end.
10. end.
11. copy all candidate 2-item sets from temp to C2.
12. temp = null.
```

3.1.2 Explanation

This step of algorithm is to generate candidate 2-item sets from whole database. In above pseudo code, n is no. of transactions; temp is temporary buffer which contains

candidate 2- item sets before finally they will be put in C_2. Whole database is first time scanned to find candidate 2-item sets. Suppose some transaction contains the items {B,C,D,F} then its candidate 2 - item sets are {B,C}, {B,D}, {B,F}, {C,D}, {C,F} and {D,F}. For each candidate 2-item set a separate count is maintained which contains the frequency of that item- set in database.

While implementing this phase, a linked list is maintained which consists of a node for each candidate 2-item set. Support count for each candidate 2-item set is calculated and stored in a field in each node. When a new candidate item-set is found a new node is inserted at the end of linked list otherwise if a node for some item-set is already there then its support count is incremented by one.

3.2 Finding Frequent 2-item Sets

Section 3.2.1 shows pseudo code to find frequent 2-item sets from candidate 2-item sets found from previous phase. That code is explained in section 3.2.2.

3.2.1 Pseudo Code

1. Repeat for each candidate 2-item set X_2
2. If (X_2.count < min_sup)
3. $C_2 = C_2 - X_2$
4. end.

3.2.2 Explanation

The candidate 2-item sets whose frequency is less than minimum requirement are removed in this phase. The set of remaining candidate 2-item sets are called frequent 2-item sets.

3.3 Generating All Possible Higher Order Candidate Item Sets

Section 3.3.1 shows pseudo code to generate higher order candidate item sets from frequent 2- item sets. That code is explained in section 3.3.2.

3.3.1 Pseudo Code

1. Repeat while (C_K is not empty and k>=2)
2. $C_{K+1} = C_K * C_K$.
3. increment k.
4. end.

3.3.2 Explanation

From previous section 3.2, no. of frequent 2- item sets is found. If conventional method like apriori is used then first frequent 2- item sets should be found from candidate 2-item sets, then from frequent 2- item sets, candidate 3-item sets are found. In general from candidate kitem sets first frequent k-item sets are found by date base scan and from frequent k-item sets, candidate (k+1) item sets are found. It is obvious that if maximum k frequent item set is possible then (k+1) database scans are to be performed. This is time consuming task.

In order to save the execution time of an algorithm, the number of database scans should be reduced with out any harmful effect on results. This can be done with scan reduction technique. From frequent 2-item sets, first candidate 3-item sets are found based on property of prior knowledge. It means for any item set to be frequent, all of its subsets must be frequent. It follows that if for some 3-item set to be frequent, all of its 2-item set subsets must be frequent. By this way all the candidate 3-item sets are found. They can not be called as frequent 3-item sets because they may not satisfy required support threshold. It will be checked with second database scan. Then from candidate 3-item sets, candidate 4-item sets are found and so on. At any time an item set X_k is put in candidate k-item set only if all its subsets are there in candidate (k-1) item sets. This process is repeated for all higher order item sets until no higher order item set is possible to generate.

In our implementation of this phase, first all candidate 3-item sets are found from frequent 2-item sets. For each one of them a new node is inserted in existing linked list of frequent 2-item sets at the end. Then candidate 4-item sets, 5-item sets etc. are found and inserted in same linked list.

3.4 Finding All Higher Order Frequent Item Sets

Section 3.4.1 shows pseudo code to find all order frequent item sets from respective candidate item sets found from previous phase. That code is explained in section 3.4.2.

3.4.1 Pseudo Code

1. Repeat for i=1 to n
2. Find all subsets having length at least 3 of item-set present in transaction i.
3. For each subset X_k in C_k
4. X_k.count = X_k.count + 1
5. end.
6. end.
7. Repeat for k=3 to m
8. Repeat for each candidate item set X_k
9. If (X_k.count < min_sup)
10. $C_k = C_k - X_k$
11. end.
12. end.

3.4.2 Explanation

This step of algorithm is to prune out infrequent item sets from any order. In this code, n is no. of transactions, m is no. of nodes in linked list. Lines 1 to 6 finds all subsets of length at least 3 for item set available in each transaction. Then support count for each subset in incremented by one in respective node. Lines 7 to 12 compare the support count for each node with minimum support required and delete the nodes whose support is less. So, after this phase, frequent item sets of all orders are present with support count in linked list.

4. Statistical Analysis

The frequent 2-base algorithm is implemented in Microsoft visual C++ 6.0. The code is executed for different size of database, different no. of items and different average transaction length. Each time min_sup is kept constant of 20. The required database is generated with pure randomness by synthetic database generator available in ARTool. All testing is done on single system which has intel core 2 duo 2.53 GHz processor, 1GB RAM and 80GB SATA HDD. In section 4.1, a comparative table is shown and briefly explained. Section 4.2 shows graphical presentation of execution time by comparing two dimensions each time and keeping value of third dimension constant.

4.1 Experimental Results

The execution time of frequent 2-base algorithm is calculated for different dimensions (parameters) like size of database (D), no. of data items (N) and average length of transaction (AT). Table (1) shows the overall results. Three different values are used for each dimension in testing. For database size D values are 100, 1000 and 10000. D=1000 means in database there are 1000 transactions available. For no. of items N values are 10, 15 and 20. N=15 means in database there are 15 different items. For average length of transaction AT values are 3, 5 and 7. All execution times are in millisecond. The values in brackets show the total no. of frequent item sets. For example, when D=10000 and N=15, if AT=3 then 27 frequent item sets are mined in 606ms, if AT=5 then 192 frequent item sets are mined in 5197ms and if AT=7 then 265 frequent item sets are mined in 23359ms.

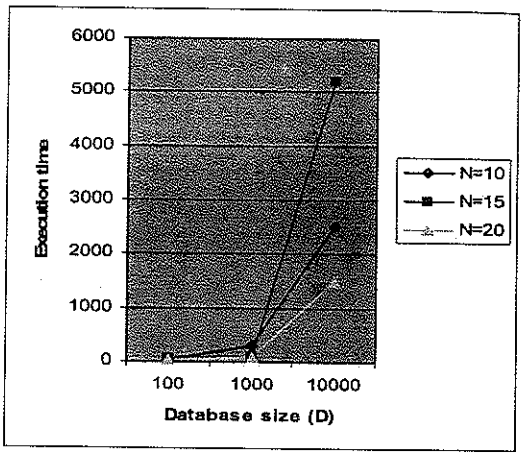
4.2 Graphial Comparision

The execution time of frequent 2-base is compared graphically for three different values of two dimensions at a time by keeping value of third dimension constant.

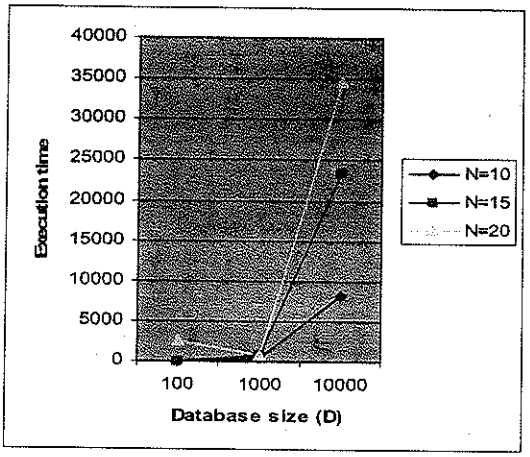
Fig.1 shows the comparison of database size and no. of items. In (a), (b) and (c) of Fig. 1 average length of transaction is kept constant of 3, 5 and 7 respectively. Fig.2 shows the comparison of database size and average length of transaction. In (a), (b) and (c) of Fig. 2 no. of items is kept constant of 10, 15 and 20 respectively. Fig.3 shows the comparison of no. of items and average length of transaction. In (a), (b) and (c) of Fig. 3 database size is kept constant of 100, 1000 and 10000 respectively.

5. Conclusion

An algorithm frequent 2-base finds all frequent item sets in two database scans. Here linked list is used as a data structure for maintaining frequent item sets. The main attraction of this algorithm is, it can be effectively used for mining frequent item sets from seasonal market basket database where some transactions or part of them are repeated over time. The general trend is that when size of database and no. of items increases, the execution time naturally increases. But it may not be the case all time. It also depends on third and most interesting parameter called average length of transaction. One can easily analyze the code of the algorithm by varying the values of all these parameters.

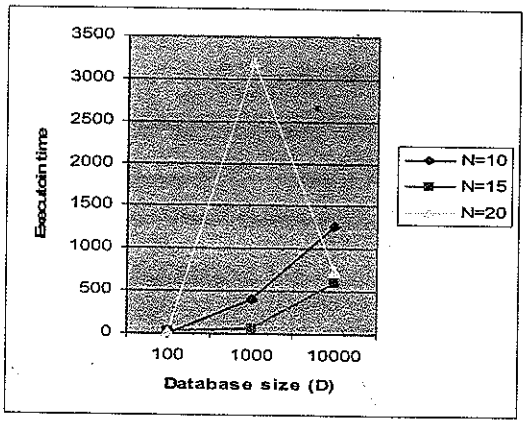


(b) AT = 5

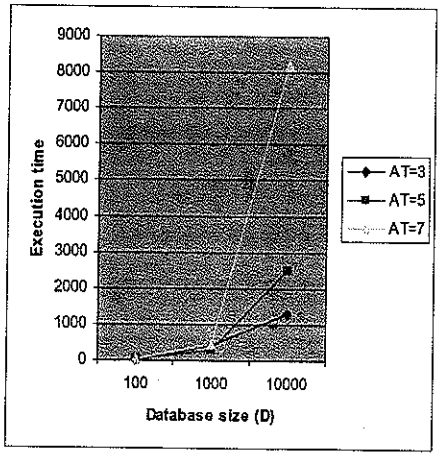


(c) AT = 7

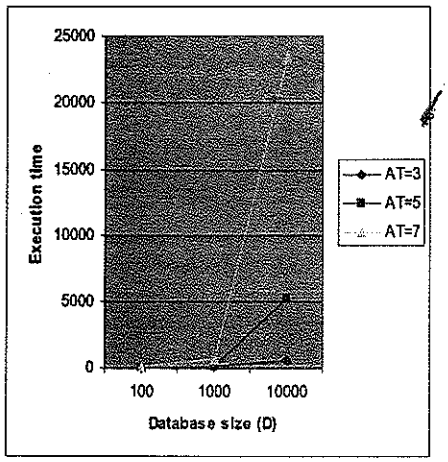
Figure 1 : Database Size Vs. No. of Items



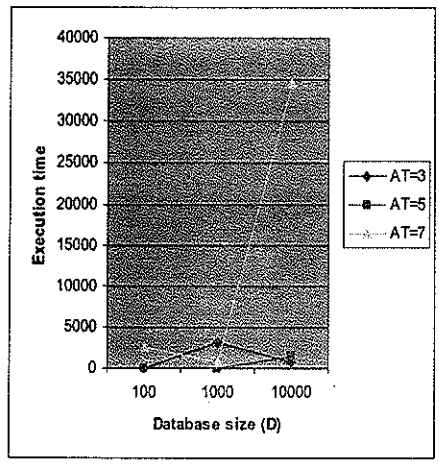
(a) AT = 3



(a) N = 10

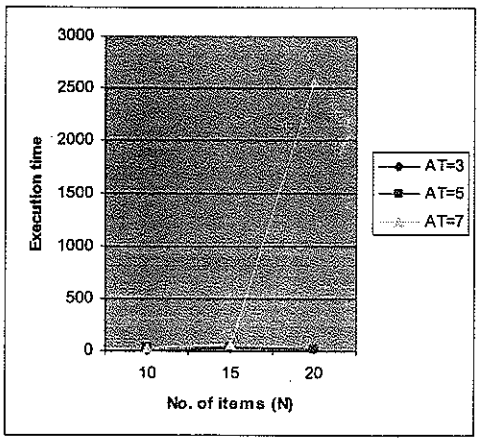


(b) N = 15

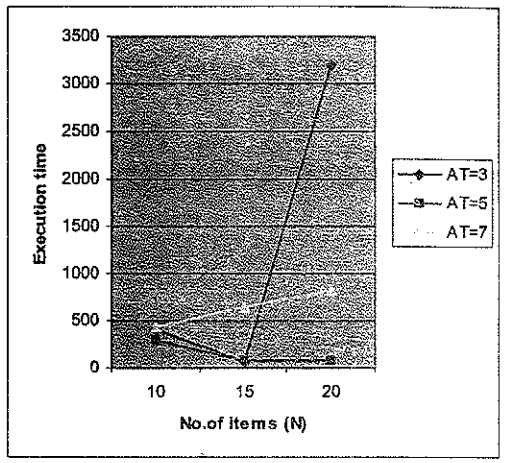


(c) N = 20

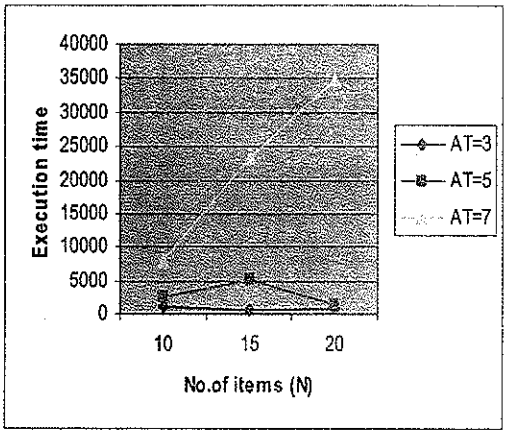
Figure 2 : Database Size Vs. Average Length of Transaction



(a) D = 100



(b) D = 1000



(c) D = 10000

Figure 3 : No. of items Vs. Average Length of Transaction

Table 1 : Execution Time (In Millisecond) For Various Values Of Dimensions

Data base size (D)	No. of Items (N)		
	10	**15**	**20**
100	0 (49), 31 (249), 31 (146)	31(108), 47(72), 62(181)	16(50), 16(34) 2562(281)
1000	406(41), 289(145), 441(285)	61(38), 77(43), 649(182)	3186(1013), 78(41), 820(208)
10000	1268(120), 506(132), 8201(336)	606(27), 5197(192), 23359(265)	731(26) 1495(50), 34727(627)

References

[1] R. Agrawal, T. Imielinski and A. Swami, *"Mining Association Rules between Sets of Items in Large Databases"*, Proc. of ACM SIGMOD, PP. 207—216, May 1993.

[2] R. Agrawal and R. Srikant, *"Fast Algorithms for Mining Association Rules in Large Databases"*, Proc. of the 20th International Conference on Very Large Data Bases, PP. 478—499, September 1994.

[3] J. Hipp, U. Güntzer and G. Nakhaeizadeh, *"Algorithms for association rule mining — a General survey and comparison"*, SIGKDD Explorations, 2(1):58—64, July 2000.

[4] Sotiris Kotsiantis, Dimitris Kanellopoulos, *"Association Rules Mining: A Recent Overview"*, GESTS International Transactions on Computer Science and Engineering, Vol.32 (1), PP. 71-82, 2006.

[5] J. Han and M. Kamber, *"Data Mining: Concepts and Techniques"*, Morgan Kaufmann Publishers, 2000.

[6] S. Brin, R. Motwani and C. Silverstein, *"Beyond market baskets: generalizing association rules to correlations"*, PP. 265—276, 1997.

[7] C. C. Aggarwal, J. L. Wolf and P. S. Yu, *"A new method for similarity indexing of market basket data"*, PP. 407—418, 1999.

[8] R. Agarwal, C. Aggarwal and V. Prasad, *"A Tree Projection Algorithm for Generation of Frequent Itemsets"*, Jornal of Parallel and Distributed Computing (Special Issue on High Performance Data Mining), 2000.

[9] J. Han, J. Pei and Y. Yin, *"Mining Frequent Patterns without Candidate Generation"*, Proc. of 2000 ACM-SIGMOD Int. Conf. on Management of Data, May 2000.

[10] J. Ale and G. Rossi, *"An Approach to Discovering Temporal Association Rules"*, ACM Symposium on Applied Computing, 2000.

[11] Junheng-huang and Wang-wei, *"Efficient Algorithm for Mining Temporal Association Rule"*, IJCSNS International Journal of Computer Science and Network Security, Vol.7 No.4, April 2007.

[12] C.H. Lee, M.S. Chen and C.-R. Lin, *"Progressive Partition Miner: An Efficient Algorithm for Mining General Temporal Association Rules"*, Accepted by IEEE Trans. on Knowledge and Data Engineering, 2002.

[13] X.Chen and I.Petr, *"Discovering Temporal Association Rules: algorithms, language And System"*, Proc. Of 200 Int. Conf. on Data Engineering, 2000.

[14] C. Aggarwal, C. Procopiuc, J. Wolf and P. Yu. Jong soo park, *"Fast algorithms for projected clustering"*, Proc. ACM SIGMOD, 1999.

[15] A. Ayad, N. El-Makky and Y. Taha, *"Incremental mining of constrained association rules"*, Proc. of the First SIAM Conference on Data Mining, 2001.

[16] C. Bettini, X. Wang and S. Jajodia, *"Mining Temporal Relationships with Multiple Granularities in Time Sequences"*, Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 1998.

Author's Biography

Prof. Mahesh H. Panchal is Head and Assistant Professor with Department of Computer Engineering, Kalol Institute of Technology & Research Centre, Gujarat. He did his graduation from Gujarat University, 2002 and

post graduation from Sardar Patel University, 2009 in computer engineering. He has a teaching experience of around 8 years. He had published a research paper in national journal, two research papers in proceeding and a paper presented in national conference. His area of interest include Database Management System, Data Mining & Warehousing, Real Time Systems.

Prof. Bhagirath P. Prajapati is Assistant Professor with Department of Computer Engineering, A. D. Patel Institute of Technology, Gujarat. He did his graduation and post graduation from Sardar Patel University 2003, 2010 respectively in computer engineering. He has a teaching experience of around 6 years. He had published a research paper in proceeding and a paper presented in national conference. His area of interest include Operating System, Windows Programming and Database Management System.