High Performance Bit Search Mining Technique

N. Venkatesan
Assistant Professor
IT Department
Bharathiyar Coll of Engg & Tech,
Karaikal

E. Ramaraj
Technology Advisor
Madurai Kamaraj University
Madurai
Tamil Nadu, India

ABSTRACT
Searching algorithms are closely related to the concept of dictionaries. String searching algorithms are too complex in all sorts of applications. To analyze an algorithm is to determine the amount of resources (such as time and storage) necessary to execute it. Most algorithms are designed to work with inputs of arbitrary length. Usually the efficiency or running time of an algorithm is stated as a function relating the input length to the number of steps (time complexity) or storage locations (space complexity). Time efficiency estimates depend on what defined to be all step. For the analysis to correspond usefully to the actual execution time, the time required to perform a step must be guaranteed to be bounded above by a constant. The main objective of this paper is to reduce the scanning the dataset by introducing new searching technique. So far, arrays, trees, hashing, depth first, breadth first, prefix tree based searching are used in association rule mining algorithms. If the size of the input is large, run time analysis of the algorithm is also increased. In this paper, a novel data structure is introduced so that it reduced dataset scan to one search. This new search technique is **bit search.** This bit search technique is to find the k^{th} itemsets (where k =1,2,3,……n) in one search scan.

Keywords
Association Rules, Breadth First Search, Depth First Search, Bit Search

1. INTRODUCTION

Finding frequent patterns plays an essential role in mining associations, correlations and many other interesting relationships among data. Moreover, it helps in data indexing, classification, clustering and other data mining tasks as well. Thus, frequent pattern mining has become an important data mining task and a focused theme in database community.

Frequent pattern mining was first proposed by Agrawal *et al*. [4] for market basket analysis in the form of association rule mining. It analyses customer buying behavior by finding associations between the different items that customers place in their shopping baskets.

Researchers have proposed several algorithms for generating frequent itemsets. These algorithms differ in their ways of traversing the itemset lattice and the ways in which they use the anti-monotone property of itemset support. Another dimension where the algorithms differ is the way in which they handle the database; i.e., how many passes they make over the entire database and how they reduce the size of the processed database in each pass.

It was proposed to address the problem of association rule mining. This is a multi-pass algorithm in which candidate itemsets are generated while scanning the database by extending known-frequent itemsets with items from each transaction. An estimate of the supports of these candidates is used to guide whether these candidates need to be extended further to produce more candidates.

Frequent itemsets are found from the dataset through several searching algorithmic approaches. Motivation of this paper is to reduce searching time by one time scanning. From this technique, k-itemset generation is found through one time matching. All the elements in a particular transaction only one time to match the k-itemset combination. This is achieved through Bit array format and its operation.

Organization of the paper is as follows: The preliminary and related works are discussed in Section 2. Section 3 describes the new algorithms and its data structure. The experimental and evaluation of new algorithms are discussed in Section 4. Performance analysis is shown in Section 5. The paper is concluded in section 6 along with concise idea on future enhancement.

2. PRELIMINARY AND RELATED WORKS

Finding frequent itemsets can be seen as a simplification of the unsupervised learning problem called "mode finding" or "bump hunting" [9] For these problems each item is seen is a variable. The goal is to find prototype values so that the probability density evaluated at these values is sufficiently large. Confidence of a rule is defined conf($X \rightarrow Y$) = supp(X U Y)/supp(X). Association rules are required to satisfy both a minimum support and a minimum confidence constraint at the same time. Since the definition of support enforces that all subsets of a frequent itemset have to be also frequent, it is sufficient to only mine all maximal frequent itemsets defined as frequent itemsets which are not proper subsets of any other frequent itemset [19].

2.1. RELATED ALGORITHMS

According to their searching nature, frequent itemset generation algorithms are classified into two categories. Breadth First Search (BFS) and Depth First Search (DFS) are two major divisions. It is further divided into two categories with counting occurrences and intersecting for both DFS and BFS. The combinations of these categories are: The first one is BFS with counting occurrence and BFS with TID List Intersection, The second is DFS with counting occurrence and DFS with TID List Intersection.
BFS with counting occurrences are the following: The most popular algorithm of this type is Apriori [4,10] where also the downward closure property of itemset support was introduced. Apriori makes additional use of this property by pruning those candidates that have an infrequent subset before counting their supports. This optimization becomes possible because BFS ensures that the support values of all subsets of a candidate are known in advance.
AprioriTID [1,2] is an extension of basic apriori approach. Instead of relying on the raw database Apriori TID internally represents each transaction by the current candidates it contains. With AprioriHybrid both approaches are combined. DIC is a further

variation of the Apriori algorithm [14]. DIC softens the strict separation between counting and generating candidates.
BFS with TID list intersections algorithm is Partition. The partition algorithm [15] is an apriori like algorithm that uses set intersections to determine support values. As described above apriori determines the support vaues of all (k-1) candidates before counting k-candidates. The problem is that partition of course wants to use the tidlists of the frequent (K-1) itemsets to generate the tidlists of the k-candidates.
DFS with counting occurrence approach algorithm is called FP Growth [6]. In preprocessing steps FP Growth derives a highly condensed representation of the transaction data, the so called FP tree, the generation of the FP-tree is done by counting occurrences and DFS. FP Growth uses the FP tree to derive the support value of all frequent itemsets.
DFS with TID List intersection algorithm [19, 20] éclat is introduced the combination of DFS with TID Lists intersections. When using DFS, it suffices to keep tid lists on the path from the root down to the class currently investigated in memory. That is splitting the databases as done by partition is no longer needed. Éclat employs an optimization, called “fast intersection”[17]. Whenever intersected with two tidlists then only intersect in the resulting tidlists. Among these algorithms is the implementation of Apriori and éclat algorithm by Borgelt [13] interfaced in the association rules environment.

2.2. BIT OPIERATIONS

A **bit stream** is a time series of bits. The term **bitstream** is frequently used to describe the configuration data to be loaded into the reconfigurable computer instead of application specific integrated circuit. A **bit array** is an array data structure that compactly stores individual bits (Boolean values). It implements a simple set data structure storing a subset of {1,2,...,*n*}. Each bit in a word can be singled out and manipulated using bitwise operations.
Although most machines are not able to address individual bits in memory, nor have instructions to manipulate single bit, each bit in a word can be singled out and manipulated using bitwise operations. For example

- OR can be used to set a bit to one: 11101010 OR 00000100 = 11101110
- AND can be used to set a bit to zero: 11101010 AND 11111101 = 11101000
- AND together with zero-testing can be used to determine if a bit is set:
  11101010 AND 00000001 = 00000000 = 0
  11101010 AND 00000010 = 00000010 ≠ 0

3. BIT SEARCH ALGORITHMS

3.1. Representation of Itemsets

From the definition of association rule mining problem that transaction databases and sets of association have in common that they contain sets of items together with additional information, For example a transaction in the database contains a transaction ID and an itemset. A rule in a set of mined association rules contain two itemsets, one for the LHS and one for the RHS, and additional quality information, ex: values for various itemset measures.

Definition 1: Transaction bit array

Let N be the number of transactions of the data set. Let M be the total number of items in the datasets. Convert the dataset items into M x N sparse matrix. Substitute all non-zero elements of sparse matrix as 1 and Mask the matrix as sparse bit matrix. Hence, keep all the transactions of the dataset as **transaction bit array**.

Definition 2: Subset bit array

Let I be a set of items. A set X = {i_1, . . . , i_k} is the subset of I is called an itemset, or a k-itemset if it contains k items. All the k-itemset are converted into bit array by substituting the presence of items as 1 and absence as 0. All subset itemsets are converted into **subset bit array**.

Definition 3 : Frequent itemset

An itemset is called frequent if its support is not less than a given absolute minimal support threshold value which is user defined one.

Definition 4: Bitwise AND

Bitwise AND operation is a novel searching technique used to find the frequent itemsets. AND can be used to find the result value for subset bit array with transaction bit array of dataset sparse bit. If the result value is as same as the subset bit array value, the k-itemsets are present in the transaction. This operation is applicable and done for all the subset k-itemsets (where k = 1,2,3,………n) and find the result in a single search. If the result value is not same as the subset bit array value, the items are not present in the transactions.

Collections of itemsets used for transaction databases and sets of association can be represented binary incidents matrices with columns corresponding to the items and rows corresponding to the itemsets .The matrix entries represents the presence (1) or absence (0) of an item in a particular itemset. An example of a binary incidence matrix containing itemsets is shown below.

Table 3.1: Sparse Matrix Representation

1	0	3	0	5
0	2	3	4	0
0	0	3	4	5
1	0	0	0	5
0	0	0	0	5

Table 3.2: Sparse Bit representation

1	0	1	0	1
0	1	1	1	0
0	0	1	1	1
1	0	0	0	1
0	0	0	0	1

Algorithm 1: Bit_Search_Item

```
1. Initialize all entries of T[I,N] as 0 as a matrix with single row for one itemset
2. N→No. of Transactions in the dataset
3. M→ No. of items in the datasets

4. // sparse bit matrix conversion

5. For k = 1 to N do
6.         For j = 1 to M do
7.                 If j = I then
8.                         S[k,j] = 1
```

```
9.              Else
10.                     S[k,j] = 0
11.             Endif
12.     End for
13. Endfor
14. // Get the k-itemsets using permutation combination or any
other procedure

15. // convert all k-itemset into bit array

16. For j = 1 to M do
17.     If j = I then
18.             T[1,j] = 1
19.     Endif
20. Endfor

21. // find the k – itemsets (k = 1,2,…..)

22. For all transaction (tid, I) € D

23. For x = 1 to k // to k-itemsets (k=1,2,…..)

24. Cx = 0 // initialize the count of k-itemsets

25. For j = 1 to M

26.     B[1,j] = s[tid,j] BITAND t[I,j]

27.     If b[I,j] = t[I,j] then
28.             Cx = Cx + 1
29.     Endif
30. End for
31. Endfor
```

Algorithm 2: Bit_Search_TID

```
1. Initialize all entries of T[I,M] as 0 as a matrix with single row
for one itemset
2. N→No. of  items in the datasets
3. M→ No. of Transactions in the dataset

4. // sparse bit matrix conversion

5. For k = 1 Mo
6.      For j = 1 to N do
7.              If j = I then
8.                      S[k,j] = 1
9.              Else
10.                     S[k,j] = 0
11.             Endif
12.     End for
13. Endfor
14. // Get the k-itemsets using permutation combination or any
other procedure

15. // convert all k-itemset into bit array

16. For j = 1 to N do
17.     If j = I then
18.             T[1,j] = 1
19.     Endif
20. Endfor

21. // find the k – itemsets (k = 1,2,…..)

22. For all transaction (tid, I) € D

23. For x = 1 to k // to k-itemsets (k=1,2,…..)

24. Cx = 0 // initialize the count of k-itemsets

25. For j = 1 to N

26.     B[1,j] = s[tid,j] BITAND t[I,j]

27.     If b[I,j] = t[I,j] then
28.             Cx = Cx + 1
29.     Endif
30. End for
31. Endfor
```

4. EXPLANATION AND EXPERIMENT

To store collections of itemsets with possibly duplicated elements (identical rows) i.e, itemsets containing exactly the same items. Since a transaction database can contain different transactions with the same items. Such a database is still a set of transactions, since each transaction also contains a unique transaction ID. The binary incidence matrix will in general be very sparse with many items and a very large number of rows. A natural representation for such data is a sparse matrix format. To implement the above procedures, the following example is used to represent the searching efficiency. Table 4.1 shows the medical details of the patients who are affected by fever, cough, throat pain, etc. with numeric transformed data items for further easy manipulation.

Table 4.1: Medical Dataset example

Symptoms	**Transformed Items**
Fever, Cough, throat pain	1 2 3 0
Fever, Cough, breathlessness	1 2 4 0
Swallowing difficulty, fever, neck swelling, Breathlessness	5 1 6 4 0
Cough, vomiting	2 7 0
Cyanosis, noisy breading, chest retraction	8 9 10 0
Cough, ear pain, ear discharge	2 11 12 0
Breathlessness, nasal block, cough, noisy breading, fever	4 13 2 9 1 0
Breathlessness, cyanosis	4 8 0

Bit_Search_Item Representation

All the transactions of the above dataset are converted into sparse matrix form and masked into sparse bit form. Table 4.2 shows the Horizontal representation of the dataset as sparse bit matrix in order to optimize process memory occupation and search time.

Table 4.2: Horizontal Sparse Bit Representation of Medical Dataset

Tid/items	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
T1	1	1	1	0	0	0	0	0	0	0	0	0	0
T2	1	1	0	1	0	0	0	0	0	0	0	0	0
T3	1	0	0	1	1	1	0	0	0	0	0	0	0
T4	0	1	0	0	0	0	1	0	0	0	0	0	0
T5	0	0	0	0	0	0	0	1	1	1	0	0	0
T6	0	1	0	0	0	0	0	0	0	0	1	1	0
T7	0	1	0	1	0	0	0	0	1	1	0	0	1
T8	0	0	0	1	0	0	0	1	0	0	0	0	0
Total	**3**	**5**	**1**	**4**	**1**	**1**	**1**	**2**	**2**	**2**	**1**	**1**	**1**

Table 4.3: Vertical representation of Medical dataset

Item/tids	T1	T2	T3	T4	T5	T6	T7	T8
1	1	1	1	0	0	0	0	0
2	2	2	0	2	0	2	2	0
3	3	0	0	0	0	0	0	0
4	0	4	4	0	0	0	4	4
5	0	0	5	0	0	0	0	0
6	0	0	6	0	0	0	0	0
7	0	0	0	7	0	0	0	0
8	0	0	0	0	8	0	0	8
9	0	0	0	0	9	0	9	0
10	0	0	0	0	10	0	10	0
11	0	0	0	0	0	11	0	0
12	0	0	0	0	0	12	0	0
13	0	0	0	0	0	0	13	0

Table 4.4: Vertical Sparse Bit Representation of Medical Dataset

Item/tids	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**	**T7**	**T8**	**Total**
1	1	1	1	0	0	0	0	0	**3**
2	1	1	0	1	0	1	1	0	**5**
3	1	0	0	0	0	0	0	0	**1**
4	0	1	1	0	0	0	1	1	**4**
5	0	0	1	0	0	0	0	0	**1**
6	0	0	1	0	0	0	0	0	**1**
7	0	0	0	1	0	0	0	0	**1**
8	0	0	0	0	1	0	0	1	**2**
9	0	0	0	0	1	0	1	0	**2**
10	0	0	0	0	1	0	1	0	**2**
11	0	0	0	0	0	1	0	0	**1**
12	0	0	0	0	0	1	0	0	**1**
13	0	0	0	0	0	0	1	0	**1**

The sparse bit representation of the table 4.2 is used to implement the Horizontal representation of association rule mining algorithms such as Apriori.

Bit_Search_TID Representation

All the transactions are first converted as item wise representation format. Table 4.3 shows the vertical representation of the dataset and its sparse matrix form.

All the itemwise transactions of the above dataset are converted into sparse matrix form and masked into transaction based sparse bit form. Table 4.4 shows the vertical sparse bit form of the medical dataset for further searching process.

The above vertical form of sparse bit representation is used to implement the association rule mining algorithms such as AprioriTID, Eclat.

All the itemsets combinations are generated through candidate key generation or any other permutation combination formula. Again the itemsets are converted into bit array structure.

From the above table 4.2, bit search is organized as the following:

All the itemsets are made ***bitwise and*** operated with the corresponding row transactions for searching itemsets by one scan. Transaction bit arrays are and operated with subset bit array. The result is compared with the corresponding itemsets bit array structure. If it is same, all the items in the itemsets are present in the transaction and count is incremented by one. Otherwise proceed to compare the next transactions and find the support level of itemset. Continue tthe above operations upto k – itemsets search in one search.

From the above table 4.4, bit search is organized as the following:

All the itemsets are made ***bitwise and*** operated with the corresponding column transaction bit arrayx for searching itemsets by one scan. Column wise bits are and operated with itemsets bit array. The result is compared with the corresponding itemsets bit array structure. If it is same, all the items in the itemsets are present in the transaction and count is incremented by one. Otherwise proceed to compare the next transactions and find the support level of itemset. Continue tthe above operations upto k – itemsets search in one search.

For example for both vertical and horizontal sparse bit representation, to search the itemsets 5,6 in the 3rd transactions the comparison is done as follows: The first 6 elements of the transaction 3 bit array is – 100111. 5,6 itemsets bit array is 000011. The comparison is 100111 bit and 000011. The result is 000011. Hence, the items are present in the transaction.

5. PERFORMANCE ANALYSIS

To prove the efficiency, all these algorithms were experimented on three data sets, which exhibit different characteristics and the results evaluated. The data sets used were: *chess, connect and mushroom* obtained from FIMI web site. For the experiments, we used Intel Pentium 2.5 GHz processor, Windows XP with 256 MB RAM was used. The results for these data sets are discussed as shown in figure 5.1 to Figure 5,3. Each figure represents the results for respective dataset implementation of Bit Search with Apriori-Trie and FP-Growth in finding k-itemset search. Diagrams are represented as the comparison of various support level and execution time which is given in seconds.

5.1 Comparison with AprioriTrie and FP-growth

Datasets are real data (Mushroom, chess and Connect-4 data) which are dense in long frequent patterns. Bit Search algorithms compared with two popular algorithms - AprioriTrie and FP-growth, the implementations of which were downloaded from http:// fimi.cs.helsinki.fi software implementation using these datasets. The characteristics of datasets are shown in Table 5.1.

Table 5.1: CHARACTERISTICS OF EXPERIMENT DATA SETS

Data	**#items**	**avg. trans. length**	**length # transactions**
mushroom	120	23	8,124
Connect-4	130	43	67,557
Chess	75	40	3,225

TABLE 5.2: RUN TIME (S) FOR CONNECT-4 DATA

Support(%)	**Bit Search**	**AprioriTrie**	**FP Growth**
5	0.969	3.531	8.984
10	0.9437	3.563	8.313
15	0.9387	3.516	8.093
20	0.9026	3.532	7.953
25	0.9008	3.516	7.734

Table 5.2 and Figure 5.1 show the relative performance of the algorithms on Connect-4 data. Connect-4 data is very dense. In the implementation Bit Search algorithm runs faster than AprioriTrie in all support level and also faster than FP-Growth.

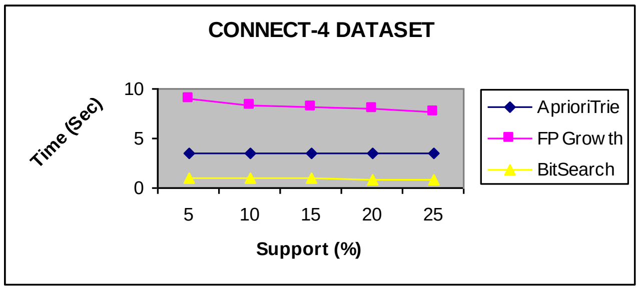

Figure 5.1: COMPARISON OF RUN TIME (S) FOR CONNECT-4 DATA

Table 5.3 and Figure 5.2 show the relative performance of the algorithms on Mushroom data. From implementation, Bit Search algorithm is faster than FP-Growth and AprioriTrie almost in all support levels.

TABLE 5.3 RUN TIME (S) FOR MUSHROOM DATA

Support(%)	**Bit Search**	**AprioriTrie**	**FP Growth**
5	0.2286	0.437	1.031
10	0.2291	0.422	0.953
15	0.2208	0.422	0.912
20	0.1998	0.422	0.902
25	0.1989	0.422	0.894

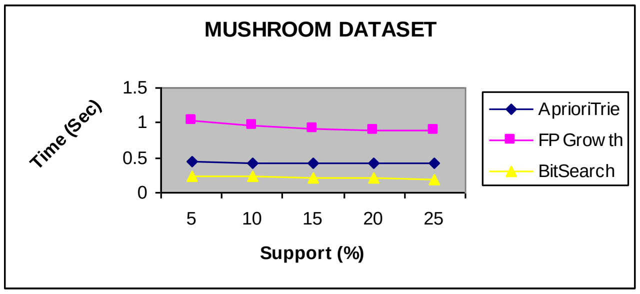

Figure 5.2: COMPARISON OF RUN TIME (S) FOR MUSHROOM DATA

Table 5.4 and Figure 5.3 show the comparison of the algorithms of interest on Chess data. AprioriTrie is better than FPGrowth while Bit Search is better than AprioriTrie.

TABLE 5.4 RUN TIME (S) FOR CHESS DATA

Support(%)	Bit Search	AprioriTrie	FP Growth
5	0.2408	0.437	1.078
10	0.2396	0.422	1.078
15	0.2213	0.422	1.031
20	0.2111	0.438	0.984
25	0.2011	0.437	0.904

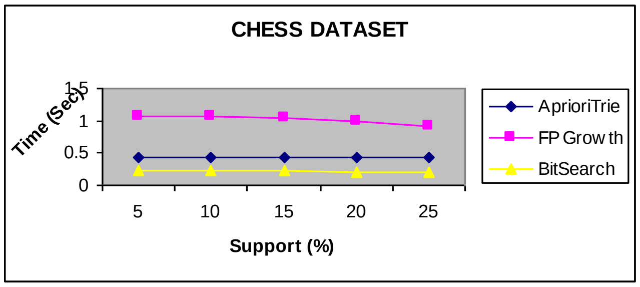

Figure 5.3: COMPARISON OF RUN TIME (S) FOR CHESS DATA

From the above diagrammatical evidence prove the efficiency of Bit Search algorithms.

In theoretical analysis of algorithms it is common to estimate their complexity in the asymptotic sense, i.e., to estimate the complexity function or arbitrarily large input. **Big O notation,** omega notation is used to the new bit search technique. Performance analysis is measured through finding the time complexity of the algorithms. Normally Turing machines are permitted bit at a time to be read or written; these are called bit operations, and the number of bit operations required to solve a problem is called its **bit complexity**. Bit complexity generalizes to any machine where to reduce the memory occupation cells are of a fixed size that depends on the input values. Put another way, the bit complexity is the complexity for all numeric values either presence or absence as a single bit.

Complexity analyses of the new proposed algorithms are defined as follows: For both horizontal and vertical sparse bit representation of the dataset are same processes.

Let number of items in dataset be M. Let the number of items in transaction be N. During the searching process, all the items in the transactions are converted as bit storage. So the required memory allocation to represent the array as blt elements. So the memory occupation is reduced. Approximate number of bytes required to the represent items and searching is calculated as **log M/2.**

Time required to search any k-itemset (k=1,2,….) in a single transaction is **1 (one)**. For N number of transaction is O(N)=N which is the lowest one while compared to any other Association Rule searching technique. In Best case, searching 1-itemset search space time is 1 and also in the worst case of k-itemset search space time is also reduced to 1. This algorithm implies its best performance for all itemset combination from 1 to k search time is reduced to **1 (one)**.

5.2. Complexity Analysis of DFS and BFS

Depth First Search Algorithm starts at a specific vertex S in G, which becomes current vertex. Then algorithm traverse graph by any edge (*u*, *v*) incident to the current vertex *u*. If the edge (*u*, *v*) leads to an already visited vertex v, then backtrack to current vertex u. If, on other hand, edge (u, *v*) leads to an unvisited vertex *v*, then go to *v* and *v* becomes our current vertex and proceed in this manner until it reaches to "deadend". Therefore, DFS complexity is O(V + E). As it was mentioned before, if an adjacency matrix is used for a graph representation, then all edges, adjacent to a vertex can't be found efficiently, that results in O(V2) complexity.

Breadth-first search is a way to find all the vertices reachable from the given source vertex, *s*. Like depth first search, BFS traverse a connected component of a given graph and defines a spanning tree. Intuitively, the basic idea of the breath-first search is this: send a wave out from source *s*. The wave hits all vertices 1 edge from *s*. From there, the wave hits all vertices 2 edges from *s*. etc. The lines added to BFS algorithm take constant time to execute and so the running time is the same as that of BFS which is O(V + E).

5.3. Advantages of Bit Search

Bit Search algorithms, despite their simplicity, have a number of marked advantages over DFS an BFS data structures:

- Bit Search algorithms are extremely compact;
- Arrays of bits to be stored and manipulated in the register set for long periods of time with no memory accesses.
- Limit memory access, and maximally use of the itemset search outperforms many other data structures on practical data sets. No other searching technique have search space time is **one.** Hence **Bit Search algorithms** are more efficient while comparing others.

6. CONCLUSION

This paper has proposed a new data structure, Bit-Search to mine frequent itemsets. Quantitative proof that Bit-Search is superior to Breadth first search (BFS) and Depth first Search (DFS) algorithms because

- reduces the search space to 1 for all itemsets combination.
- Reduces the memory space for finding the frequent itemsets.

- Increases the efficiency
- Decrease the time complexity.

The advantages of Bit-Search over existing searching algorithms listed above are good evidence for efficiency. Bit-Search scores a scalable height to implement in association rule mining algorithms especially when transactions are large. By evidence, bit search technique of vertical representation execution time is compared with FP Growth and AprirorìTrie. Bit search has low execution time. This new technique will be applied in any type of searching related algorithms as future extension work.

7. REFERENCES

[1] Zhi-Choa Li, Pi-Lian He, Ming Lei, "A High Efficient Aprio,iTID Algorithm for mining Association rule", Proceedings of 4th International Conference on machine learning and cybernetics, pp 18-21 AUG 2005.

[2] He Li-jian, Chen Li-chao, Liu shuang-ying, "Improvement of AprioriTid Algorithm for Mining Association Rules", Journal of Yantai University, Vol.16, No.4, 2003.

[3] R. Agrawal, J.Shafer, "Parallel mining of association rules", IEEE Transactions on knowledge and Data Engineering, 8(6), December 1996.

[4] R. Agrawal, T. Imielinski, and A.N. Swami. Mining association rules between sets of items in large databases. In P. Buneman and S. Jajodia, editors, Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, volume 22(2) of SIGMOD Record, pages 207–216. ACM Press, 1993.

[5] Ke Su, Fengsdhan Bai *Mining weighted Association Rules* IEEE transactions on KDE 489-495, April 2008

[6] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," Procedings of ACM SIGMOD Intnational Conference on Management of Data, ACM Press, Dallas, Texas, pp. 1-12, May 2000.

[7] J. Pei, J. Han, H. Lu, S. Nishio, S. Tang, and D. Yang, "Hmine: Hyper-structure mining of frequent patterns in large databases," Proc. of IEEE Intl. Conference on Data Mining, pp. 441-448, 2001.

[8] A. Pietracaprina, and D. Zandolin, "Mining frequent itemsets using Patricia Tries," FIMI '03, Frequent Itemset Mining Implementations, Proceedings of the ICDM 2003 Workshop on Frequent Itemset Mining Implementations, Melbourne, Florida, December 2003.

[9] G. Grahne, and J. Zhu, "Efficiently using prefix-trees in mining frequent itemsets," FIMI '03, Frequent Itemset Mining Implementations, Proceedings of the ICDM 2003 Workshop on Frequent Itemset Mining Implementations, Melbourne, Florida, December 2003.

[10] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. Proceedings 20th International Conference on Very Large Data Bases, pages 487–499. Morgan Kaufmann, 1994.

[11] C. Borgelt and R. Kruse. Induction of association rules: Apriori implementation. In W. H¨ardle and B. R¨onz, editors, Proceedings of the 15th Conference on Computational Statistics, pages 395–400, http://fuzzy.cs.unimagdeburg.de/~borgelt/software.html2002. Physica-Verlag.

[12] D. Burdick, M. Calimlim, and J. Gehrke, "MAFIA: a maximal frequent itemset algorithm for transactional databases," Proceedings of International Conference on Data Engineering, Heidelberg, Germany, pp. 443-452, April 2001,

[13] Christian Borgelt *Efficient implementation of Apriori and Eclat FIMII '*03

[14] Bart Goethals *Survey on Frequent Pattern Mining,* , HIIT Basic Research Unit, University of Helsinki, Finland.

[15] Ja-Hwung Su, Wen-Yang Lin: CBW: An efficient algorithm for Frequent Itemset Mining, Proceedings of 37th Hawaii International Conference on System Science 2004.

[16] Data Mining – Concepts and Techniques, Jiawei Han, Micheline Kamber – 2004 Edn

[17] Mingju Song and Sanguthevar Rajasekaran *A transaction mapping for frequent itemsets mining* IEEE transactions on Knowledge and Data Engineering 18(4):472-480, April 2006.

[18] Balaji Padmanabhan, Alexandar Tuzhilin *On Characterization and Discovery of Minimal unexpected pattern in Rule Discovery* IEEE trans on KDE vol 18 no. 2 Feb '06

[19] M.J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li, "New algorithms for fast discovery of association rules," Proceedings of the Third International Conference on Knowledge Discovery and Data Mining, AAAI Press, pp. 283-286, 1997.

[20] M.J. Zaki, and K. Gouda, "Fast vertical mining using diffsets," Proceedings of the Nineth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, D.C., ACM Press, New York, pp. 326-335, 2003.

[21] P. Shenoy, J. R. Haritsa, S. Sudarshan, G. Bhalotia, M. Bawa, and D. Shah, "Turbo-charging vertical mining of large databases," Procedings of ACM SIGMOD Intnational Conference on Management of Data, ACM Press, Dallas, Texas, pp. 22-23, May 2000,

Authors

Dr. E. Ramaraj is presently working as a Technology Advisor, Madurai Kamaraj University, Madurai He has 24 years teaching experience and 14 years research experience. He has presented research papers in more than 50 national and international conferences and published more than 30 papers in National and International journals. His research areas include Data mining and Network security.

N.Venkatesan is working as an Assistant Professor, Information Technology Department, Bharathiyar College of Engineering and Technology, Karaikal. He has 13 years of teaching experience. He has been member of ISTE. He published 4 papers in International journals 12 and papers in National and International conferences. He has authored a book ***Data Mining and Warehousing***.