BIT SEARCHING TECHNIQUE

N. Venkatesan*

E. Ramaraj**

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

* Research Scholar.

** Technology Advisor, Madurai Kamaraj University, Madurai.

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*. [1] 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. Problem definition is described in the section 3. Section 4 describes the new algorithms and its data structure. The experimental and evaluation of new algorithms are discussed in Section 5. Performance analysis is shown in Section 6. The paper is concluded in section 7 along with concise idea on future enhancement.

2. RELATED WORK

EXISTING SEARCHING TECHNIQUES OF ARM

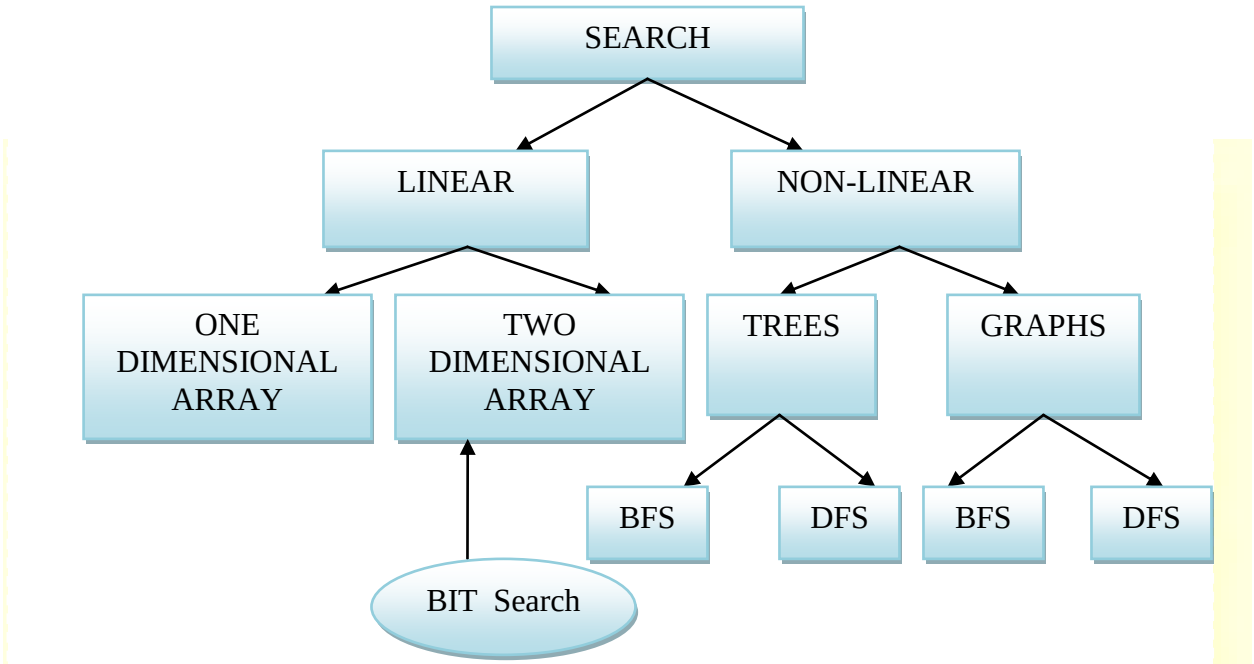

Figure 2.1: Types of Searching Technique

The search technique is classified into two major categories in data structure [5] [6]. First one is known as Linear and second is called non-linear. Figure 2.1 shows the classification of searching techniques. Linear search includes one dimensional and two dimensional arrays. Trees and Graphs come under non-linear search. According to Trees and Graphs searching nature, algorithms are classified into two categories. Breadth First Searcch (BFS) and Depth First Search (DFS) are two major divisions. Figure 2.2 represents the categories of Association rule mining algorithms with respect to BFS and DFS implementation. BFS and DFS are the two search techniques that play a vital role in frequent itemset generation. 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.

.

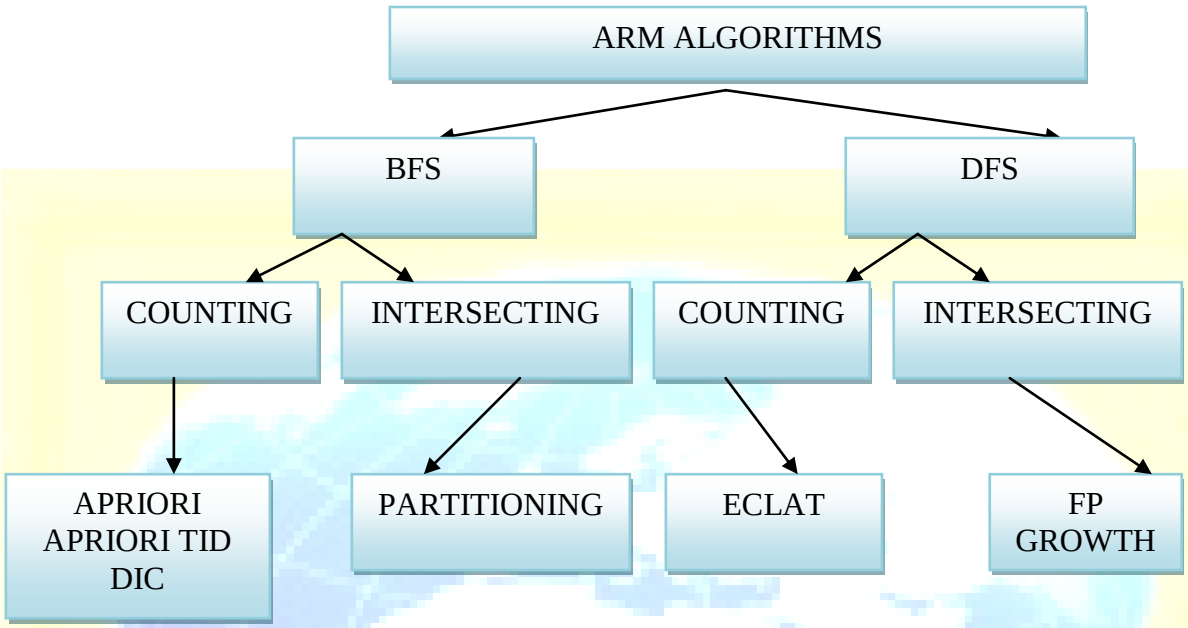

Figure 2.2: Categories of Association Rule Mining Algorithms

The most popular algorithm of this type is Apriori [1] [2] 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 [3] 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 DIC softens the strict separation between counting and generating candidates.

BFS with TID list intersections algorithm is Partition. The partition algorithm is an apriori like algorithm that uses set intersections to determine support values. As described above apriori determines the support values 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. 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 éclat is introduced by 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". Among these algorithms is the implementation of Apriori and éclat algorithm by Borgelt [4] interfaced in the association rules environment.

3. PROBLEM DEFINITION

Data Mining has more techniques to mine useful pattern from the dataset. One such technique is Association rule generation, is done with the help of frequent itemsets. Frequent itemsets are generated with the help of searching of items in the given dataset. Dataset contains transaction and items. More than one item makes the itemsets. Searching the itemsets are done with many types of algorithms. Major research is to reduce the searching time of the algorithms. Bit search

is a novel search procedure which generates frequent itemset which overcome this problem in order to make association among items of the datasets. Bit search is used to find frequent itemsets.

4. BIT SEARCH ALGORITHMS

4.1. 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 $X = \{i_1, \ldots, i_k\}$ *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 4.1: 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 4.2: 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
5. // 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
```

5. 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 5.1 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 5.1: 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 5.2 shows the Horizontal representation of the dataset as sparse bit matrix in order to optimize process memory occupation and search time.

Table 5.2: Horizontal Sparse Bit Representation of Medical Dataset

Tid/ite	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 5.3: 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 5.4: 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 5.2 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 5.3 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 5.4 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 5.2, 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 5.4, 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.

6. PERFORMANCE ANALYSIS

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. The bit search 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 is M and let the number of items in transaction be N. During the searching process, all the items in the transactions are converted as bit storage. The required memory allocation is to represent the array as bit elements. So the memory occupation was 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).

7. CONCLUSION AND FUTURE ENHANCEMENT

Finding frequent itemset generation is an important task in forming association rules. Itemsets are searched with the help of data structure. Data structure searching techniques are classified into linear and non linear types. Linear type is divided into one dimensional and two dimensional arrays. Non linear type is separated by trees and graphs. Breadth First Search and Depth First Search are used in trees and graphs for searching process. A new Bit search technique is introduced for searching itemsets in two dimensional arrays with the help of transaction bit array, subset bit array and Sparse Bit matrix. Bit AND operation is used to find the k-itemsets matching in one search. New Association Rule Mining applications are developed using this new search technique.

REFERENCE:

1. Agrawal, R., Imielinski, T., and Swami, A. N. 'Mining association rules between sets of items in large databases'. In Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, 1993, pp: 207-216.
2. 10. Aumann, Y. and Y. Lindell, ,A statistical theory for quantitative association rules'. Proceedings of the 1999 International Conference on Knowledge Discovery and Data Mining, Aug. 15-18, 1999. San Diego, CA., USA., pp: 261-270.
3. 77. Zhi-Choa Li, Pi-Lian He, Ming Lei, 'A High Efficient AprioriTID Algorithm for mining Association rule', Proceedings of 4th International Conference on machine learning and cybernetics, pp 18-21 AUG 2005
4. Christian Borgelt 'Efficient implementation of Apriori and Eclat' FIMI 2004
5. Sahni Sartaj, Data Structures, Algorithm, and Applications in C++, McGrawal Hill College di . 1998.
6. Ellis Horowitz, Fundamentals of Data Structures in C++, W.H. Freeman and Co. 1995
7. Gade, K., J. Wang and G. Karypis, 'Efficient closed pattern mining in the presence of tough block constraints'. Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Aug. 22-25, 2004. Seattle, WA., pp: 138-147.
8. Geerts, F., B. Goethals and J. Bussche, 'A tight upper bound on the number of candidate patterns'. Proceedings of the 2001 International Conference on Data Mining, Nov. 29-Dec. 2, 2001. San Jose, CA., pp: 155-162.
9. Grahne, G. and J. Zhu, ,'Efficiently using prefix-trees in mining frequent itemsets'. Proceedings of the 2003 ICDM International Workshop on Frequent Itemset Mining Implementations, (IWFIMI'03), 2003. Melbourne, FL., pp: 123-132.
10. Han, J., J. Pei, Y. Yin and R. Mao, 'Mining frequent patterns without candidate generation: A frequent-pattern tree approach'. Data Mining Knowledge Discovery, 2004.8: 53-87
11. Hidber, C., 'Online association rule mining'. ACM SIGMOD Rec., 28: 1999. PP: 145-156.

12. Srikant V and Agrawal, R. 'Fast Algorithms for Mining Association Rules', Proc. VLDB Conference, 1994, pp 487–499.

13. Li, Z., Chen, Z. Srinivasan S.M. and Zhou, Y. 'C-Miner: Mining block correlations in storage systems'. Proceedings of the 3rd USENIX Conference on File and Storage Technologies, March 31, 2004. San Francisco, CA., pp: 173-186.

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