Improving Efficiency of Apriori Algorithm using Cache Database

Priyanka Asthana
VIth Sem, BUIT, Bhopal
Computer Science Deptt.

Divakar Singh
Computer Science Deptt.
BUIT, Bhopal

ABSTRACT

One of the most popular data mining approach to find frequent itemset in a given transactional dataset is Association rule mining. The important task of Association rule mining is to mine association rules using minimum support value which is specified by the user or can be generated by system itself. In order to calculate minimum support value, every time the complete database has to be scanned for each item in the transaction. This decreases the time complexity of the algorithm. Here we proposed a new algorithm which scan the database once and create a cache database for each transaction using hash map. This cache copy is then used to search for frequent item sets. Due to which the overhead of scaning complete database for each item is reduced, and efficiency is increased.

Key word: Apriori, cache database, hash map, scanning time, time complexity.

1. INTRODUCTION

1.1. Association Rule Mining

Association rule mining is the efficient method which is used in finding the association rules[8]. The key to find the association rules is to find all the frequent item sets present in the given transactional record by means of the minimum support threshold.

Let I={i1, i2,. . ., im} be a set of items and D be a set of transactions, where each transaction T (a data case) is a set of items so that T $\subseteq$ I. An association rule [12] is an implication of the form, X --> Y, where X $\subseteq$ I, Y $\subseteq$ I and X $\cap$ Y= ϕ. The rule X-->Y holds in the transaction set T with confidence c, if c% of transactions in T that support X also support Y. The rule has support s in T if s% of the transactions in T contains X $\cup$ Y. In a database D, given a set of transactions , the problem of mining association rules is to discover all association rules that have support and confidence greater than the user-specified minimum support (called minsup) and minimum confidence (called minconf).

The key element that makes association-rule mining practical is minsup. This is used to prune the search space and to limit the number of rules generated. However, when only a single minsup is used, it implicitly assumes that all items in the database are of the same nature or of similar frequencies in the database. This is not the case in real-life applications [3, 4]. In the retailing business, customers are suppose to buy some items very frequently but other items very rarely. Usually, the necessities, consumables and low-price products are bought frequently, while the electric appliance, luxury goods and high-price products infrequently. In this situation, if minsup is too high, all the observed patterns are concerned with those low-price products, which only contribute a small portion of the profit to the business. On the other hand, if minsup too low, too many meaningless frequent patterns will be generated and they will overload the decision makers, who may find it difficult to understand the patterns generated by data mining algorithms.

The dilemma faced in the above application is called the rare item problem [5]. In view of this, researchers either (A) split the data into a few blocks according to the frequencies of the items and then mine association rules in each block with a different minsup [6], or (B) group a number of related rare items together into an abstract item so that this abstract item is more frequent [6,7]. The first approach is not satisfactory because rules that involve items across different blocks are difficult to find. Similarly, second approach is unable to find rules that involve individual rare items and the more frequent items. Clearly, both approaches are adhoc and "approximate" [6].

To solve the above said problem, Liu et al. [3] have extended the existing association rule model to allow the user to specify multiple minimum supports to reflect different natures and frequencies of items. Specifically, user can specify a different minimum item support for each item. Thus, different rules may be needed to satisfy different minimum supports depending on what items are in the rules. This new model named Apriori with time slice, enables users to produce rare item rules without causing frequent items to generate too many meaningless rules. However, the proposed algorithm named MSapriori algorithm in Liu et al. [3], adopts an Apriori-like candidate set generation-and-test approach and it is always costly and time-consuming, especially when there exist long patterns.

To solve this problem, systematic algorithm [1] was proposed in which user is not allowed to specify any minimum support threshold values to find the frequent patterns; instead the system itself generates the minimum threshold values, therefore plugging the loophole of other algorithms. This algorithm also introduces the concept of timing algorithm along with the systematic algorithm, which will statically assign a unique value to each record of the transactional database. This algorithm is mainly used to save time by scanning through the entire transactional database only once rather than making multiple scans. The profit of one scan database leads to better performance and minimization of time. In this study, we propose a novel cache database structure, which extends the hashing Apriori algorithm [2] to store binary sring for each transactional record in the file as a index. The experimental result shows that the algorithm is efficient, and that it is about an order of magnitude faster than the apriori algorithm.

1.2. Hash method

A hash function is any algorithm or subroutine that maps data sets of variable length to data sets of a fixed length. Hash functions are mostly used to quicken table lookup or data comparison tasks such as finding items in database, discover duplicated or similar records in a large file and so on.

Hash functions are primarily used in hash tables, to quickly locate a data record given its search key. Specifically, the hash function is used to map the search key to an index; the index gives the place in the hash table where the corresponding record should be stored. A hash function [9], h is a function which transforms a key from a set, K, into an index in a table of size n. Following is the hash function:

h: K -> {0, 1, ..., n-2, n-1}

1.2.1. Direct Address Tables

If we have a collection of n elements whose keys are unique integers in (1,m), where m >= n, then we can store the items in a *direct address* table, T[m], where either T_i is empty or contains one of the elements of our collection. Following Fig 1. shows direct access table.

Searching a direct address table is clearly an O(1) operation:

For a key, **k**, we access $\mathbf{T_k}$,

- if it contains an element, return it,
- if it doesn't then return a NULL.

There are two constraints here:

1. the keys must be unique, and
2. the range of the key must be severely bounded.

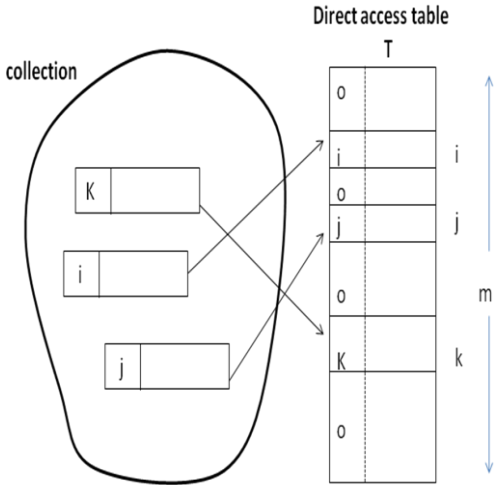


Figure 1. Direct Access Table

1.2.2. Mapping Functions

The direct address approach [9] requires that the function, h(k), should be a one-to-one mapping from each k to integers in (1,m). Such a function is known as a perfect hashing function: it maps each key to a distinct integer within some manageable range and enables us to trivially build an O(1) search time table.

Unfortunately, finding a perfect hashing function is not always possible. Let's say that if there is a hash function, h(k), which maps *most* of the keys onto unique integers, but small number of keys are mapped on to the same integer. If the number of collisions (cases where multiple keys map onto the same integer), is very small, then *hash tables* work quite well and give O(1) search times.

2. RELATED WORK

2.1 Basic Apriori

Apriori is a classic algorithm for frequent itemset mining and association rule for transactional databases [10]. This algorithm identifies the frequent individual items in the database and extending them to larger and larger item sets as long as those item sets appear frequent in the database. Apriori gives frequent itemsets which can be used to determine association rules which accent general trends in the database: this has applications in domains such as market basket analysis. Apriori is a "bottom up" approach, where frequent itemsets are considered one item at a time, and groups of itemsets(candidates) are tested against the database[13]. The algorithm terminates when further successful extensions are not found.

Since Apriori algorithm was first introduced and as experience was accumulated, there have been many attempts to find more efficient algorithms of frequent itemset mining [11]. Many of these share the same idea with Apriori in that they generate candidates.

2.2 Apriori with time slice algorithm

In this algorithm[1], the user is not allowed to specify any minimum support threshold values to find the frequent patterns; instead the system itself generates the minimum threshold values, thus removing the drawback of other algorithms. This algorithm also introduce the concept of timing algorithm along with the systematic algorithm, which will statically assign a unique value to each record of the transactional database. Mainly this technique is used to reduce time by scanning through the entire transactional database only once rather than making multiple scans. This algorithm takes any dataset as input, and a systematic table is constructed for every transaction provided in the dataset.

2.2.1 Systematic Algorithm [1]:

The systematic tables for every itemsets involved in the datasets are calculated by the following conditions:

Supp(A--->B) = supp(A)+ supp(B)+ supp(AUB)

Supp (A--->¬B) =supp (A) - supp (A UB)

Supp (¬A--->B) =supp (B) - supp (A UB)

Supp (¬A---> ¬B) =1- supp (A) - supp (B) + supp(A UB)

2.2.2 Timing Algorithm [1]:

T: For each of the itemsets in TID do

Find the count of a pattern as

Count (1, TDB) = {(transid, x) (transid, x)}

Milepost of Negative Support:

Supp-^{n} = n /σ ((S^{n} (1)), Where $1 \leq n \leq$ Count (1)

Milepost of positive support:

Supp+ ^{n} = [Count (1) - n) / [TD - σ (S^{n} (I))],

Where $1 \leq n \leq$ Count (1)

The benefit of one scan database gives better performance and minimize total time.

2.3 Hash based Apriori method

Hash based Apriori method, uses a data structure that directly represents a hash table [2]. This algorithm overcome some of the weaknesses of the Apriori algorithm by reducing the number of candidate k-itemsets. In particular the 2-itemsets, since that is the key to improving performance. This algorithm uses a hash based technique to reduce the number of candidate itemsets generated in the first pass. It has proved that the number of itemsets in C2 generated using hashing can be reduced, so that the scan required to determine L2 is more efficient.

3. ISSUES IN FINDING ASSOCIATION RULES

During the process of searching from the database, the entire database is scanned more than once or only once. This scanning of the entire database at least once also create problem.

1. Firstly, searching for items in the database through the entire database may increase the search space complexity. A lot of memory is needed for each search of the database.

2. Secondly, searching through the entire database may increase the time taken to find the required item also.

3. Security of database.

To overcome above said issues, this paper proposes a new algorithm called Apriori with cache database in which for searching it uses Hash Map which uses the Binary String for the creation of the CahedCopy of the Database.

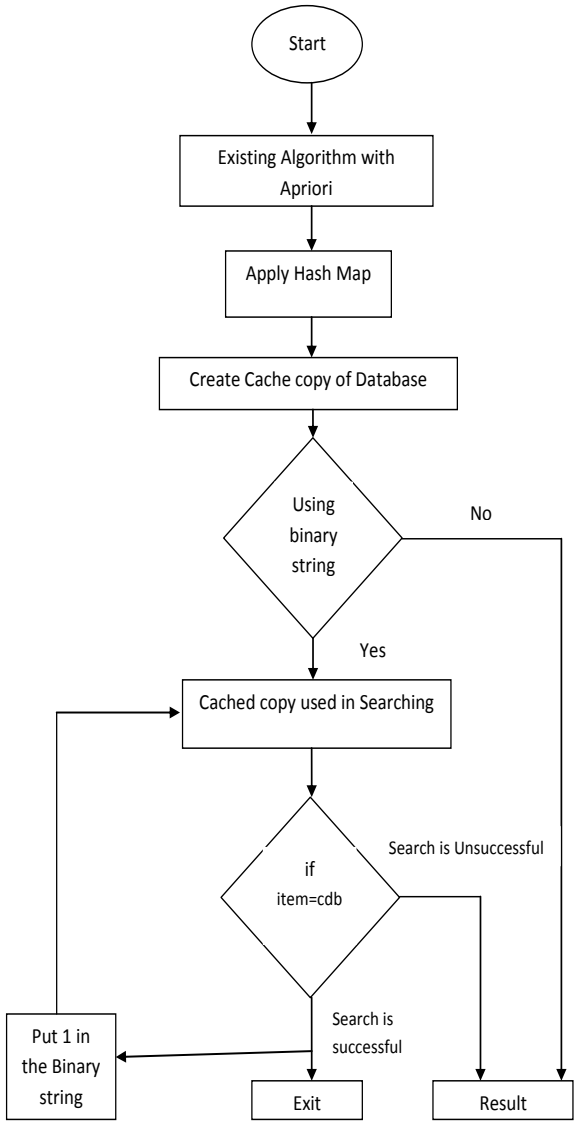
3.1. Our Contribution

In this paper, there is a algorithm called Apriori with cache database, which can reduce the time complexity of apriori algorithm, by scanning the whole database once. This algorithm creates a cache database which store all frequent itemsets found by apriori algorithm. Then generate binary string for each transactional database and store it in a file which behave as a hash index. So every time now for candidate generation, we will search this cache database for itemset instead of the whole database. The experimental result shows that our method is effective efficient. This algorithm reduce the scaning time, which in turn increase the efficiency of the apriori algorithm.

4. PROPOSED WORK

This algorithm "Apriori with Cache Database" (Apriori CDB) works on the fast Hashing technique. It hashes the entire database and puts it into a software Cache and from where the retrieval is very easy. The algorithm uses the naïve method (Apriori Algorithm) to find the frequent Item set. It reads the data set over and over in every iteration to find the frequent item set for string of different lengths. But the searching method is diferent, which make our algorithm more efficient. For searching this method use Hash Map which uses the Binary String for the creation of the CahedCopy of the Database. By this binary string it can easily search item set with more accuracy with less time. Another advantage of this is, if this method is applied on large database that will give more accurate results with less time complexity in comparison of existing algorithm. Following is the flow chart of proposed algorithm(Fig. 2)

Figure 2. Flow chart of proposed algorithm



Proposed Hashing method : Instead of reading the database again and again it goes through the database once and create Hash map for the individual element in the database .Hash map <String,Int> maps the item to an integer. But here it uses a bit different method to create a hash map. It creates a Binary string for per line(or per transaction) in the database so our Cached Database is a database of binary strings .

Consider the following database (Table 1) for the Caching. First of all Frequent itemsets are found by using basic Apriori Algorithm. Now this table include the entries of all frequent itemsets generated by Apriori algorithm. The algorithm for the creation of the binary string per line is as follows:

Table 1: Assumed database

TID	List of Items
T1	I1, I2, I5
T2	I2, I4
T3	I2, I3
T4	I1, I2, I4
T5	I1, I3
T6	I2, I3
T7	I1, I3
T8	I1, I2, I3, I5
T9	I1, I2, I3

From the above database (Table 1), Then for every line in database we create a binary string of length N. In our cache database, we have key values as a index. Now read every line and then we find the location of the items in the line and put a "1" in the binary string corresponding to that location. Below (Table 2) is our Cache Database showing binary strings generated for each transactions.

Table 2: Our Cached database

Index	Binary String
T1	10011
T2	11110
T3	00101
T4	01010
T5	10000
T6	01110

Now Every time when we have to search for any item in the database for generation of 2-itemsets, instead of searching it in the main database, we will search it from our cache database. Therefore after each joining step, a hash map is created from where searching of item is done. While pruning items are retrieved from cache database. As soon as item is found, scanning is not required for rest transactions (rows). Therefore scanning time also reduces here. Thus, there is no need to scan the original database for searching. The creation of binary string also provide security.

4.1 Steps of Proposed Algorithm (Apriori CDB)

Call procedure of (A-priori)

1. Create Hash Map for individual element in the database .
2. Apply bit different method in hash map.
3. Create cache copy of data base (CDB) .
4. Apply binary string (BS) for per line in the database.
5. Store binary string(BS) into Cache data base(CDB) .
6. Arranging item sets on the basis of binary string in hash table.
7. Use Cache data base (CDB) for item set search.
8. Calculate the value of N (No of records (lines) in Data base).
9. Initialize variable set Count =0 and item Defines the records of which you want to search ;
10. While (count <=EOF)
11. Count = count +1;
12. N = count;
13. End
14. If(item = CDB)
15. Search is Successful and put 1 in the binary string correspond to that location.
16. Store new BS into CDB.
17. Else Unsuccessful.
18. Exit.

5. EXPERIMENTAL RESULT

In order to evaluate the efficiency of the Apriori CDB algorithm such as times for searching items in transaction databases we choose several size of databases. Our method gives efficient result on large dataset. We are comparing our experimental result with my base paper, Apriori with time slice algorithm [1].

The database consists of all frequent itemsets generated by Apriori Algorithm in first iteration, having minimum support value 1. We are giving experimental results of "Apriori with time slice algorithm" [1] and my proposed algorithm "Apriori with cache database" on different-different size of data sets.

Table 3: Comparison of execution time when MS=1

S. No	Size of Data Set(KB)	Execution time 0f Apriori with time slice algo (in millisecs)	Execution time of Apriori with cache database (in MilliSec)
1.	1	27371	26178
2.	2	25695	23333
3.	3	27376	24006
4.	4	28383	27296
5.	19	57088	36955

From the above table, we can see that the execution time of our proposed method (Apriori CDB) takes less time as compared to Apriori with time slice. As our data size increases our algorithm takes much lesser time as compared to Apriori with time slice. This result also shows that our algorithm gives better result as the size of dataset increased.

The following figures 3 shows the comparison of time of both algorithms (Apriori CDB and Apriori with time slice) in millisecond using graph when MS=1. Data size in KB is taken in X-axis, and time in millisecond is taken in Y-axis.

Time also varies when minimun support value is changed from 1 to 2. When we set minimum support value=2, instead of 1, result shows that again execution time of Apriori CDB is less than the execution time of Apriori with time slice.

Following is Table 4 shows execution time of both algorithm when MS=2:

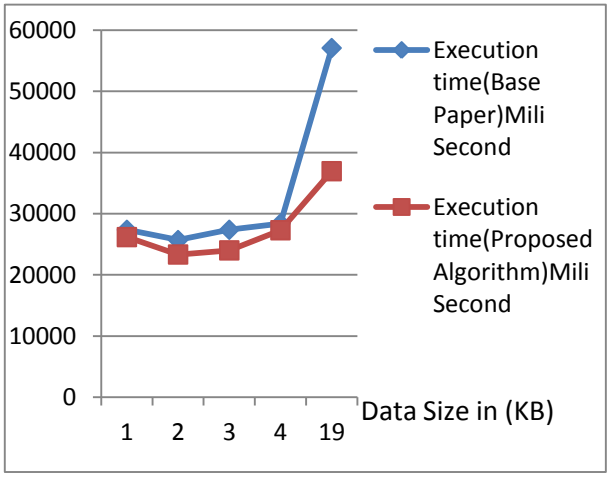


Figure 3. Comparison of time in millisecond when MS=1

Table 4: Comparison of execution time when MS=2

S.No.	Data set size in (KB)	Execution time of Apriori with time slice algo (in millisecs)	Execution time of Apriori with cache database (in MilliSec)
1	2	42459	40914
2	4	73437	72470
3	10	75586	72307
4	19	76280	76859
5	66	116727	97683
6	110	269608	168882

Above table shows, when we set MS value 2, again execution time of our algorithm is less than the Apriori with time slice algorithm. As datasize increases, time complexity of our algorithm reduces as compared to Apriori with time slice algo.

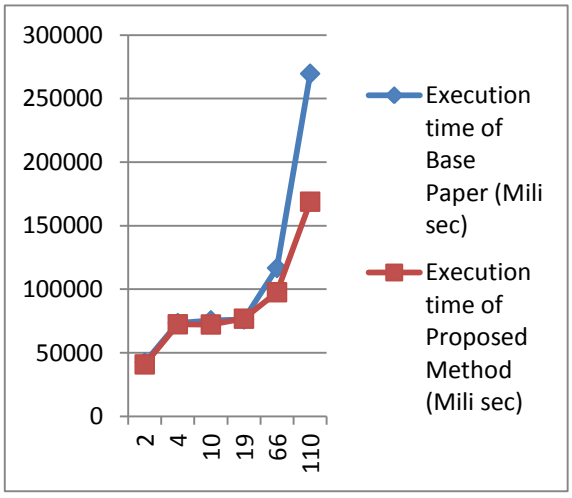


Figure 4. Comparison of time in millisecond when MS=2

Figures 4 shows the comparison of time of both algorithms (Apriori CDB and Apriori with time slice) in millisecond using graph when MS=2. Data size in KB is taken in X-axis, and time in millisecond is taken in Y-axis.

When we set minimum support value=3, instead of 2, result shows that again execution time of Apriori CDB is less than the execution time of Apriori with time slice. Following is Table 5 shows execution time of both algorithm when MS=3:

Table 5: Comparison of execution time when MS=3

S.No.	Data set size in (KB)	Execution time of Apriori with time slice algo (in millisecs)	Execution time of Apriori with cache database (in MilliSec)
1	2	42509	40964
2	4	73487	72520
3	10	75636	72357
4	19	76330	76909
5	66	116777	97733
6	110	269658	168932

Following figures 5 shows the comparison of time of both algorithms (Apriori CDB and Apriori with time slice) in millisecond using graph when MS=2. Data size in KB is taken in X-axis, and time in millisecond is taken in Y-axis.

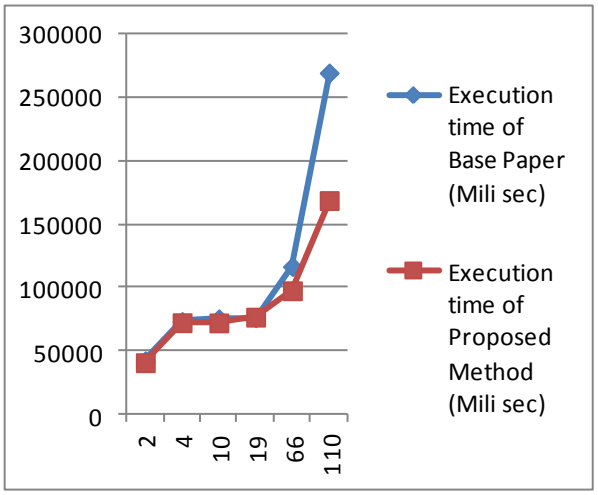


Figure 5. Comparison of time in millisecond when MS=3

6. CONCLUSION AND FUTURE SCOPE

Time is the major factor in real life applications. This algorithm has reduced the time complexity of Apriori Algorithm using cache database. Table 3 shows execution time of both the algorithm. Every time for every data size, my proposed algorithm gives better result as compared to Apriori with time slice algorithm. Our result also shows, if we apply our method on large database (bigger data size), that will give more accurate results with less time complexity. We presented experimental results, showing that the proposed algorithm always outperform Apriori with time

slice. The effectiveness of our algorithm is shown experimentally and practically.
Further research can be done on time and space complexity, combined with some other techniques to reduce space and time complexity.

7. REFERENCE

[1] S.Sangeetha, "Verdict of Association Rule Using Systematic Approach of Time Slicing for Efficient Pattern Discovery " Proceeding of 2012 International Conference on Computing, Electronics and Electrical Technologies [ICCEET].

[2] K.Vanitha and R.Santhi, "Using Hash Based Apriori Algorithm to Reduce the Candidate 2- itemsets for Mining Association Rule "Proceeding of H.S.Behera et al, Journal of Global Research in Computer Science, Volume 2, No 5, 2011.

[3] B. Liu, W. Hsu, Y. Ma, Mining association rules with multiple minimum supports, Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-99), San Diego, CA, USA, 1999.

[4] M.C. Tseng, W.Y. Lin, "Mining generalized association rules with multiple minimum supports", International Conference on Data Warehousing and Knowledge Discovery (DaWaK'01), Munich, Germany, 2001, pp. 11 – 20.

[5] H. Mannila, "Database methods for data mining", Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining (KDD '98) tutorial, New York, NY, USA, 1998.

[6] W. Lee, S.J. Stolfo, K.W. Mok, "Mining audit data to build intrusion detection models", Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining (KDD '98), New York, NY, USA, 1998.

[7] J. Han, Y. Fu, "Discovery of multiple-level association rules from large databases", Proceedings of the 21th Very Large DataBases Conference (VLDB'95), Zurich, Switzerland, 1995, pp. 420– 431

[8] Ya -Han Hu and Yen-Liang Chen, "Mining Association rules with multiple minimum Supports: a new mining algorithm and a Support tuning mechanism", Elsevier B.V. All rights reserved, Decision Support Systems, 42, (2006) pp.1-24.

[9] http://www.arl.wustl.edu/projects/fpx/cse535/lecture/cse535_lecture6_Hash_Functions.pdf

[10] http://en.wikipedia.org/wiki/Apriori_algorithm.

[11] XindongWu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, Dan Steinberg, "Top 10 algorithms in data mining", © Springer-Verlag London Limited 2007.

[12] R. Agrawal. T. Imielinski. and A Swami, "Mining Association Rules between Sets of Items in Large Databases", Proc. 1993 ACM SIGMOD Int'I Conf. Management of Data (SIGMOD '93), pp. 207-216, 1993.

[13] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," Proc. 20th Int'I Conf. Very Large Data Bases, pp.487-499, 1994.