An Efficient Distributed Algorithm for Computing Association Rules

Yijun Li[1] Xuemin Lin[1] Chi-Ping Tsang[2]

[1] School of Computer Science and Engineering, University of New South Wales Sydney, NSW 2052, Australia. Email: {yijunli, lxue}@cse.unsw.edu.au
[2] Department of Computer Science, University of Western Australia Nedlands, WA 6907, Australia. E-mail: tsang@cs.uwa.edu.au

Abstract. Data mining aims to efficiently discover previously unknown knowledge from large databases. It is highly demanding in numerous real-life applications, such as marketing strategy, financial forecast, etc. One of the fundamental problems in the area is the efficient computation of association rules. In this paper, we shall investigate this problem in a distributed database. Particularly, we will present an efficient distributed algorithm for mining distributed association rules. Our experiment results suggest that the proposed algorithm outperforms the existing distributed algorithms. Further, we also study a distributed version of the problem of association rules; and extend our algorithm to solve this new problem.

Keywords: Relational Databases, Distributed Data Mining, and Algorithms.

1 Introduction

The discovery of "association rules" in databases may provide useful background knowledge to decision support systems, selective marketing, financial forecast, medical diagnosis, and many other applications. The problem of efficiently computing association rules has received a great deal of attention in the past several years; and a number [2, 6, 9, 11] of efficient algorithms have been proposed to approach this problem. As the size of a database to be mined can be very large, parallel computation techniques have also been explored [1, 10].

Consider that in a distributed organization, the database may be allocated through a computer network. This leads to a real demand for developing distributed computation techniques in data mining. In this paper, we shall restrict ourself to an investigation of mining association rules in a distributed database.

In [9], an efficient distributed algorithm is proposed. It should be clear that the parallel algorithms developed in [1, 10] can be immediately used as distributed algorithms. Comparing the distributed algorithm DMA [4] with an implementation of the parallel algorithm CD [1] in a distributed environment, DMA is more efficient than CD because a reduction of both network communication and local processing costs has been made in DMA. The experiment results in [4] confirmed such an improvement.

Although DMA has a smaller message overhead than that of an implementation of the parallel algorithm PDM [10] in a distributed environment, our initial experiment results showed that in a fast network such as LAN, PDM is faster than DMA in many cases. A careful analysis of DMA and PDM reveals that PDM is based on a faster local processing algorithm (sequential algorithm) DHP [9], while DMA employs a better network communication protocol. This leads to a consideration of combining together the advantages from both DMA and PDM into one algorithm. Further, according to our recent research results, there exists a faster sequential algorithm, PLO-DHP, than DHP. The PLO-DHP enjoys a speed-up of up to 50% over DHP according to the experiment results [6]. In this paper, we will present a new distributed algorithm based on PLO-DHP, and it employs a standard polling technique in distributed computing as suggested in DMA. Our experiment results suggest that our algorithm is much more efficient than both PDM and DMA in a fast network, and has a low message overhead. This is the first contribution of the paper.

The second contribution of the paper is to extend our algorithm to solve a new problem - mining *distributed association rules*; that is, computing association rules that are shared by at least N component databases where N is defined by users. For instance, in a multi-national supermarket company association rules shared by its N local supermarkets will give us the information of common purchase interests of local residents, so that good decisions of goods transportation and organization cross the company can be made. We call the problem "N common association rules problem"; and it will be formally defined in section 2.

In fact, the association rules problem is a special case of the distributed association rules problem (N common association rules problem). Specifically, when $N = 1$, these two problems are equivalent. This yields a simplification of the presentation of our results in the paper. The above two contributions will be combined into a single algorithm - MNA (stands for **M**ining of N common **A**ssociation rules). Considering the limitation of paper length, in this paper we present only the algorithm of mining "N commonly large itemsets", which is the most expensive and important part in mining N common association rules, while the interested readers may refer to [7] for the other details.

The rest of paper is organized as follows. In section 2, we will formally define the problem, and present necessary preliminaries. In section 3, we present the algorithm - MNA. Section 4 gives the experiment results and discussions. This is followed by conclusions.

2 N Common Association Rules

Let $I = \{i_1, i_2, ..., i_m\}$ be a set of literals, called *items*. Suppose that there are n sites $\{s_i : 1 \leq i \leq n\}$ in the computer network. At each site s_i, D_i is a set of transactions (tuples), where each transaction T is a set of items (that is, $T \subseteq I$), and is identified by its local ID L_TID. The global database D is the union of the component databases D_i where each tuple T is identified by its local ID together by the site ID i (that is, identified by (L_TID, i)). Following the assumptions made in [2, 9, 11], we assume that the items in each transaction (tuple) are sorted on their keys in an increasing order. A transaction T *contains* a set X of items in I, if $X \subseteq T$.

An *association rule* is an implication of the form $X \rightarrow Y$, where $X \subseteq I$, $Y \subseteq I$, and $X \cap Y = \emptyset$. The rule $X \Rightarrow Y$ has support s in the database D if $s\%$ of transactions in D contain $X \cup Y$. The rule $X \Rightarrow Y$ holds in the transaction set D with confidence c if $c\%$ of transactions in D that contain X also contain Y.

With respect to a given minimum support value $s\%$, a given confidence $c\%$, and a given integer $N \leq n$, the problem of mining N *common association rules* is to find all association rules $X \Rightarrow Y$ in D, such that

> $X \Rightarrow Y$ has support s and confidence c in D as well as in at least N component databases in D.

A set X of items is a *k-itemset* if it consists of k items, and is *large* if there are at least $s\%$ of transactions in D that contain X. A k-itemset is *N commonly large* if it is large in D as well as large in at least N component databases of D.

Following the framework [2] for mining association rules, we are going to solve the problem by two phases: 1) compute N commonly large itemsets; and then 2) determine the N common association rules from the large itemsets.

The computation of the second phase is trivial and inexpensive, and thus will not be presented in this paper. The interested readers may refer to [7] for the details. In this paper, we mainly discuss the first phase.

Below, we present the notation used in the rest of the paper. Suppose that a distributed database $D = \cup_i^n D_i$ is given where each D_i is component database that reside at site i; and $s\%$, $c\%$, and N are all given. We use $L_{k,N}$ to denote the set of k-itemsets that are N commonly large in D, and use $C_{k,N}$ to denote a superset of $L_{k,N}$. Note that when $N = 1$, $L_{k,1}$ and $C_{k,1}$ are abbreviated to L_k and C_k. We use L_k^i to denote the set of k-itemsets that are large in the component database D_i.

Like the other developments [1, 4, 10], our algorithm for computing N commonly large itemsets will follow the framework [2] - Algorithm Apriori, and be conducted in an iterative fashion:

Algorithm Apriori
Scan D to obtain L_1
for $(k = 2; L_{k-1} \neq \emptyset; k++)$ **do**
  { $C_k \leftarrow gen(L_{k-1})$; //generate candidates by L_{k-1}.
    if $C_k \neq \emptyset$ **then**
      { $count(C_k)$; //count candidates against D.
        $L_k \leftarrow \{c \in C_k | c.count \geq s\% \times |D|\}$; }
    else $L_k \leftarrow \emptyset$ }
Return $\cup L_k$.

Although the algorithm DHP [9] follows the same iterations as suggested in

Apriori, in its first iteration a hash table H_2 is built while L_1 is generated. Then in the second iteration of DHP, we check C_2 against H_2 to reduce the size of C_2. Further, in the end of the second iteration of DHP, the original database is trimmed to a smaller one for later iterations. Experiment results [9, 6] showed that DHP is more efficient than Apriori.

In our recent research paper [6], we presented an improvement of DHP that has been proven more efficient than DHP. This algorithm is named as PLO-DHP. Our distributed algorithm MNA is based on PLO-DHP.

3 Algorithm MNA

In this section, we present an efficiently distributed algorithm, NMA, for mining N common association rules. The algorithm MNA employs the algorithm PLO-DHP as a base mining algorithm, and applies a standard distributed technique - polling site, as suggested in [4].

Suppose that a distributed transaction database $D = \{D_i : 1 \le i \le n\}$ is given such that the n component databases D_is reside at n different sites. Further, let the minimum support value $s\%$, confidence $c\%$ and N be given. The following theorem provides a fundamental base to the correctness of the algorithm MNA; and the interested readers may refer to [7] for a detailed proof.

Theorem 1. *For any given N and k, $L_{k,N} \subseteq \cup_{i=1}^{n} L_k^i$.*

The algorithm MNA iteratively executes the following local processing and message communication phases.

Phase 1: Local Processing. In order to avoid multiple scans of local databases at an iteration k, each site i will generate the global $C_{k,N}$ and do counting on $C_{k,N}$ against its local database to produce $LC_k^i = L_k^i \cap C_{k,N}$.

Phase 2: Message Communication. Once LC_k^i is generated at each site i, message communication through the network is required to generate $L_{k,N}$; and then $L_{k,N}$ will be broadcasted to all sites for the next iteration. In the second iteration, network communication is also required for generating H_2.

Let H_2^i denote the hash table used in DHP for mining D_i. The algorithm MNA is presented below.

Algorithm MNA

M1: Each site i generates L_1^i and H_2^i by scanning
    D_i;
M2: $L_{1,N} \leftarrow COMP(\cup_{i=1}^{n} L_1^i)$;
M3: $H_2 \leftarrow COMP(\cup_{i=1}^{n} H_2^i)$;
M4: **while** $(|L_{k-1}| \ge k)$ **do**
M5: { **if** $(k = 2)$ **then** at each node i **do**
M6: { $C_{2,N} \leftarrow MNA\_gen(L_{1,N})$;
M7: $LC_2^i \leftarrow MNA\_count(C_{2,N})$;

M8: $D'_i \leftarrow MNA\_trim(D_i)$;
M9: $L_{2,N} \leftarrow COMP(\cup_i LC_2^i)$; }
M10: **else** at each node i **do**
M11: { $C_{k,N} \leftarrow MNA\_gen(L_{k-1,N})$;
N12: $LC_k^i \leftarrow MNA\_count(C_{k,N})$;
M13: $L_{k,N} \leftarrow COMP(\cup_i LC_k^i)$; }
M14: $k \leftarrow k + 1$; }
M15: **Return** $\cup_k L_{k,N}$.

In MNA, line M1 is to generate all locally large 1-itemsets and to build a hash table for each local database D_i, and it is the same as the fisrt iteration in DHP [10]. The lines M6, M7, M8, M11 and M12 are in the phase of local processing and will be explained in Section 3.1, while the lines M2, M3, M9 and M13 are in the phase of message communication and the details will be given in Section 3.2.

3.1 Applying Triple Pruning and On-Line Indexing

As noted [6, 9], the local processing costs in mining large itemsets are determined by sizes of databases and generated candidates. Smaller the sizes, lower the costs. Motivated by this, in the algorithm we employ a triple pruning technique to reduce the size of candidates, as well as applying the on-line indexing technique [6] to logically and vertically truncate the database.

A Triple Pruning Technique In [2], a sort-merge join on L_{k-1} and L_{k-1} is proposed that generates C_k from L_{k-1}, and together with a pruning procedure that removes an element c from C_k if a sub $(k-1)$-itemset of c is not in L_{k-1}.

In [4], it suggested that to take the advantage of distributed *data pattern skew*, we should include only the elements c in C_k such that all sub $(k-1)$-itemsets of c are locally large at one common component database. This is the second pruning technique applied in our algorithm.

Consider that in many real applications, the size of C_2 is much larger than the sizes of other candidate sets. In [9], a hash pruning technique is provided for reducing the size of C_2. To do this, firstly a hash table H_2 is built; and then each element $c \in C_2$ is checked against an entry H_2 to determine if it should be pruned from C_2.

We noted that these three pruning techniques can be modified to be used in mining N commonly large itemsets. Moreover, in our algorithm MNA we propose to combine these three techniques together to form a triple pruning technique. Let c be a k-itemset and $S(c)$ denote the set of common sites at which all sub $(k-1)$-itemsets of c are locally large. Now we present our algorithm to generate $C_{k,N}$ from $L_{k-1,N}$. Suppose that the items in each element of $L_{k,N}$ are stored according to the increasing order of item keys.

$\underline{MNA\_gen(L_{k-1,N})}$:

$C_{k,N} \leftarrow L_{k-1,N} \bowtie L_{k-1,N}$; //two tuples from $L_{k-1,N}$ joins on the first $k-2$ items.
for each element $c \in C_{k,N}$ **do**
  if $(\exists c' \subset c \wedge |c'| = k-1 \wedge c' \notin L_{k-1,N})$ **then** $C_k \leftarrow C_k - \{c\}$;
  else { compute $S(c)$; **if** $(|S(c)| < N)$ **then** $C_k \leftarrow C_k - \{c\}$; }
Return C_k.

Note that when $k = 2$ each $c \in C_2$ also needs to be checked against H_2 to determine whether or not it should be included in C_2; this results in a triple pruning technique. More details about hash pruning technique will be presented in Section 3.2 when we introduce message communication phase of the algorithm MNA. Further, the procedure MNA_gen $(L_{k-1,N})$ is implemented in such a way that once a c is generated from $L_{k-1} \bowtie L_{k-1}$, the pruning criteria shall be checked immediately to determine if c should be included in C_k will be given in [7].

The full details of the proof that for $k \geq 1$, $C_{k,N} \supseteq L_{k,N}$ while applying the procedure MNA_gen This guarantees the generation correctness of candidates.

An On-line Bitmap Indexing Technique At each iteration, once the global candidates $C_{k,N}$ have been obtained, each site i will do local counting against its database D_i. Besides building up a hash tree index T_k on $C_{k,N}$ to speed up the counting process, in [6] we proposed an on-line bitmap indexing technique to logically reduce the length of each tuple. The idea is quite simple. For each tuple $T \in D_i$, instead of using T for counting $C_{k,N}$, we need only to use $T' \subseteq T$ where T' consists only items that are contained by elements in $C_{k,N}$. To implement this efficiently, an on line bitmap index $B_k[l]$ (for $0 \leq l \leq |I|$) is created in the main memory, such that $B_k[l] = 1$ if and only if the item i_l is contained in an element in $C_{k,N}$. The paper [6] gives the detailed implementation of the technique; and the experiments report that there will be up to 50% reduction on counting costs by applying this technique comparing with the standard counting process adopted in Apriori and DHP.

Therefore, the algorithm MNA will adopt the on-line bitmap indexing technique combining with the standard counting process in [2, 9].

3.2 Applying Polling to Reduce Message Overhead

After counting $C_{k,N}$ at each site i, every element c in $C_{k,N}$ has a count c_i in D_i. To assemble the global count for each element $c \in C_{j,N}$ from its local counts, a network communication must be invoked. To minimize the message overhead, a polling technique was proposed [4]. In our algorithm, we will apply this technique to both procedures $COMP(\cup_i LC_j^i)$ and $COMP(\cup_i H_2^i)$.

Note that in the procedure of $COMP(\cup_i H_2^i)$, a hash bucket in an H_2^i is created corresponding to the value of $H(x, y)$ where H is a hash function and

(x, y) is a 2-itemset. Each hash bucket stores the number of 2-itemsets falling into it. At each site i, let HL_2^i denote the subset of hash buckets in H_2^i whose counts are larger than $s\% \times |D_i|$. Besides, a same hash function h is employed at each site to hash HL_2^i, and h has n hash buckets. The procedure $COMP(\cup_i H_2^i)$ consists of the following steps:

Step 1: At each site i, hash the buckets of HL_2^i using h. If the hash value is l then send the corresponding hash bucket together with its count to site l. Let $HL_2(l)$ denote the hash buckets that site l received from all other sites.

Step 2: At each site l, once it receives all messages from the other sites it will assemble the global count for each element c in $HL_2(l)$ if c is from at least N sites. To assemble the global count for c, site l needs only to contact the sites j where c is not in HL_2^j.

Step 3: Let $HL_2'(l)$ denote the set of buckets in $HL_2(l)$ whose global count is not less than $s\%|D|$. At each site i, broadcast $HL_2'(i)$ to the other sites; and then build up H_2 once it receives all $HL_2'(j)$ from the other sites j; H_2 gives the binary information of whether the global count of a hash bucket obtained using hash function H is less than $s\%|D|$.

$COMP(\cup_i LC_j^i)$ is implemented in a similar way. Next we present an example to explain the algorithm MNA.

3.3 An Illustration of MNA

In this subsection, we first use one example to illustrate the algorithm MNA. Suppose that there are three sites together with three component databases, as depicted in Figure 1. Let $N = 2$ and $s = 40$.

Assume that we define the hash function as

$$H(x, y) = \{(\text{order of } x) \times 10 + \text{ order of } y\} \bmod 7.$$

After the first iteration, $L_{1,2} = \{A, B, C, D, F\}$, and the global hash table is illustrated in Figure 2. Note that:

$$\begin{aligned} L_{1,2} \bowtie L_{1,2} = \{&(A, B), (A, C), (A, D), (A, F), \\ &(B, C), (B, D), (B, F), (C, D), \\ &(C, F), (D, F)\}. \end{aligned}$$

Note that the pruning technique as stated in "if" phase of MNA_gen is not applicable in the second iteration. If we use the pruning technique in the "else" phase of MNA_gen, (A, B) and (A, F) will be removed from $L_{1,2} \bowtie L_{1,2}$. If we use the hash pruning technique as described in MNA, the 2-itemsets (B, D), (C, F), and (D, F) will be eliminated from $L_{1,2} \bowtie L_{1,2}$. Therefore, a combination of these two pruning techniques, as suggested in our algorithm MNA, will remove $\{(A, B), (A, F), (B, D), (C, F), (D, F)\}$ from $L_{1,2} \bowtie L_{1,2}$, that is, there are only five elements left in $C_{2,2}$ using MNA.

Site1

L_TID	Items
1	A, C, E
2	A, D, F
3	B, C, D

Site2

L_TID	Items
1	A, B, C, D, F
2	A, C, D
3	B, F

Site3

L_TID	Items
1	A, B, C, D
2	B, C, F
3	D, F

Fig. 1. An example database

Using the network communication method introduced in section 3.2, we obtained that $L_{2,2} = \{(A, C), (A, D), (B, C), (C, D)\}$. Using MNA, $C_{3,2} = \emptyset$. Thus, the algorithm MNA terminates.

The Theorem 1 and the fact that $C_{k,N} \supseteq L_{k,N}$ guarantee that the algorithm MNA is correct.

4 Experiment Results and Discussions

For performance evaluation purpose, the algorithm DMA, the algorithm PDM, and our algorithm MNA have been implemented on a 10M shared nothing LAN, which connects four Pentium/133. The four connected PCs run Windows NT system; and each of them has 64M main memory. We implement these algorithms using Java/RMI. The database was generated using the benchmark generation software downloaded from the web site of IBM Almaden Research Center, and the software uses the parameters as shown in Figure 3

In our experiments, to generate a distributed database, we first use the benchmark generation software to generate $|D|$ and then randomly partion $|D|$ into four component database D_i for $1 \leq i \leq 4$. The component databases are respectively stored in the local disks.

In the first experiment, we choose $M = 1000$, $|I| = 4$, $|D| = 400,000$, and $|T| = 15$. Besides, we choose $30,000$ as the bucket number to build a hash table H_2, and choose 300 as the number of children in each hash tree index

Bucket_id	Count	2-itemsets contained
0	5	(A, D), (C, E)
1	3	(A, E), (C, F)
2	6	(A, F), (B, C)
3	3	(B, D)
4	3	(D, F)
5	5	(A, B), (B, F)
6	8	(A, C), (C, D)

Fig. 2. H_2

|D|	Number of transactions
|T|	Average size of transactions
|I|	Average size of the maximal potentially large itemsets
|L|	Number of potentially large itemsets
M	Number of items

Fig. 3. Parameters

on candidate sets. In this experiment, the algorithm MNA runs the parameter $N = 1$; this corresponds to the problem of mining association rules.

The experiment results are depicted in Figure 4. From Figures 4 (c) and (d), we can see that the algorithm MNA is faster than both PDM and DMA. There are two interesting phenomena occurred - while the minimum support value goes smaller,

Ph1: a reduction effect on the execution time of BMA by MNA is decreasing, and

Ph 2: a reduction effect on the execution time of PDM by MNA is increasing.

Note that while the minimum support value goes smaller, more large itemsets will be formed, and thus, the effect of local processing cost reduction by PLO-DHP is reduced. This is why Ph1 occurs. Further, while the minimum support value goes smaller, more candidates will be generated, and thus network communication costs are increased. Since DHP applies a broadcast communication protocol, the communication costs increase in DHP is much lager than that in MNA. This is why Ph2 occurs. We should also note that the execution time of MNA may be

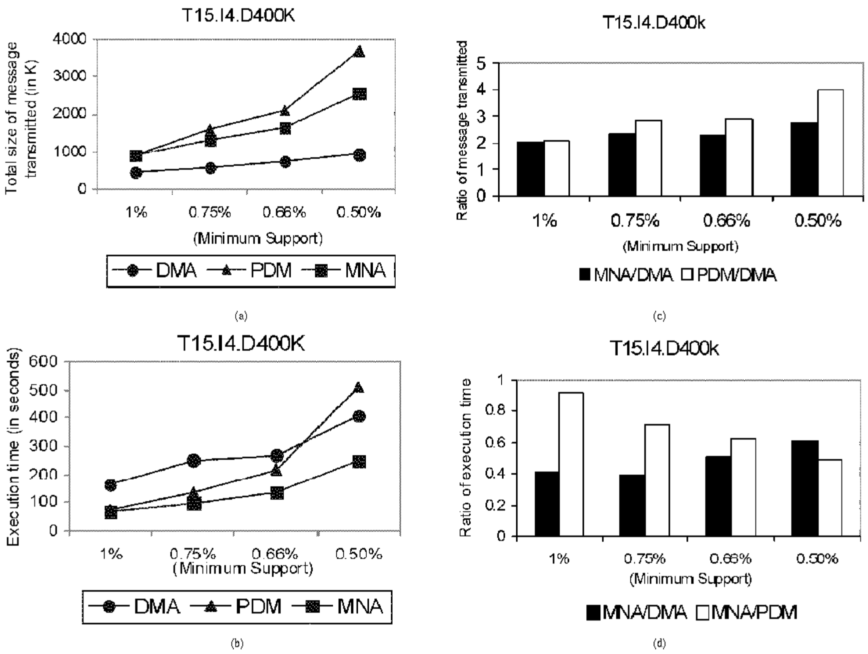

Fig. 4. Results of the experiment 1

as less as 50% of that of PDM and 40% of that of DMA.

The experiment shows that the total message size of MNA is larger than that of DMA. This is because an extra demand of network communication for building H_2 happens in the iteration two of MNA. However, the total message size of MNA is smaller than that of PDM. This is because that the communication protocol used in MNA is better than PDM in a distributed environment.

In our second experiment, we try to show the speed up caused by an increase of N. The experiment results are shown in Figure 5, where $M = 1000$ and $|D|$ ranges from $100,000$ to $1,000,000$.

Note that both DMA and PDM can also be naively extended to solve the N common association rules. While applying DMA when $N > 1$, the number of candidates generated in each iteration is smaller than that in the corresponding iteration for $N = 1$. This is also true for MNA and PDM. Therefore, given a minimum support value s and $N > 1$, the time relationship among DMA, PDM, and MNA should be similar to that among DMA, PDM, and MNA for a larger value s_0 (i.e., $s_0 > s$) and $N = 1$. This means that MNA should be most efficient as well when $N > 1$.

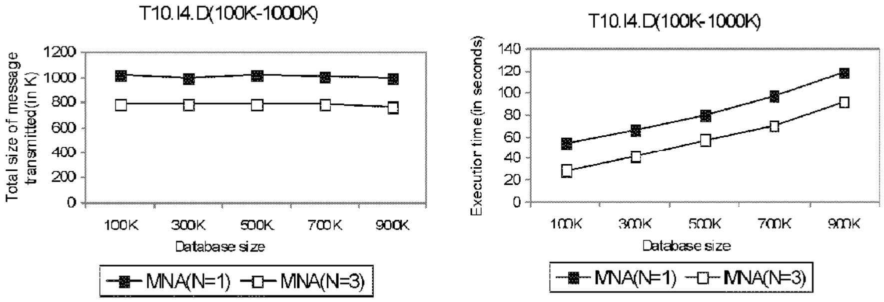

Fig. 5. Results of the experiment 2 ($s\% = 0.5\%$)

5 Conclusion

In this paper, we studied the problem of mining distributed association rules, a generalized problem of mining association rules. We proposed an efficiently distributed algorithm MNA to solve this problem. Our experiment results suggest that if restricted to the association rules problem, the algorithm MNA is more efficient than the existing algorithms DMA and PDM.

We recently also developed Web access pattern mining algorithms based on PLO-DHP [6]. As a future study, we will develop efficient distributed algorithms to solve Web data mining problem.

References

1. R. Agrawal and J. C. Shafer, Parallel Mining of Association Rules: Design, Implementation, and Experience, *IEEE Transactions on Knowledge and Data Engineering*, 8(6), 962-969, 1996.
2. R. Agrawal and R. Srikant, Fast Algorithms for Mining Association Rules, *Proceedings of the 20th VLDB Conference*, 487-499, 1994.
3. R. Agrawal and R. Srikant, Mining Sequential Patterns, *Proceedings of the 11th International Conference on Data Engineering*, 1995.
4. D. W. Cheung, Vincent T. Ng, Ada W. Fu and Y. Fu, Efficient Mining of Association Rules in Distributed Databases, *IEEE transactions on Knowledge and Data Engineering*, 8(6), 884-897, 1996.
5. U. M. Fayyad, G.Piatetsky-Shapiro, P.Smyth, and R.Uthurusamy. *Advances in Knowledge Discovery and Data Mining*, AAA/MIT Press, 1996.
6. X.Lin , Y.Li and C.P.Tsang, Applying On-Line Bitmap Indexing to Reduce Counting Costs in Mining Association Rules, *Information Science*, 120 (1-4), 197-208, 1999.
7. Y.Li, On Efficient Computation of Association Rules, *Master Thesis*, University of New South Wales, 2000.
8. R. Ng and J. Han, Efficient and Effective Clustering Method for Spatial Data Mining, *Proc Int'l Conf. Very Large Data Bases*, 144-155, Santiago, Chile, Sept, 1994.

9. J.S. Park, M.S. Chen, and P.S. Yu , An Effective Hash-Based Algorithm for Mining Association Rules, *Proc. ACM-SIGMOD Int'l Conf. Management of Data*, 175-186, 1995.
10. J.S. Park, M.S. Chen, and P.S. Yu, Efficient Parallel Data Mining for Association Rules, *Proc. Int'l Conf. Information and Knowledge Management*, 1995.
11. A. Savasere, E.Omiecinski, and S.Navathe, An Efficient Algorithms for Mining Association Rules in Large Databases, *Proc. Int'l Conf. Very Large Data Bases*, 432-444, Zurich, Sept. 1995.