A SURVEY OF ASSOCIATION RULES

Margaret H. Dunham, Yongqiao Xiao
Department of Computer Science and Engineering
Southern Methodist University
Dallas, Texas 75275-0122

Le Gruenwald, Zahid Hossain
Department of Computer Science
University of Oklahoma
Norman, OK 73019

ABSTRACT: Association rules are one of the most researched areas of data mining and have recently received much attention from the database community. They have proven to be quite useful in the marketing and retail communities as well as other more diverse fields. In this paper we provide an overview of association rule research.

1 INTRODUCTION

Data Mining is the discovery of hidden information found in databases and can be viewed as a step in the knowledge discovery process [Chen1996] [Fayyad1996]. Data mining functions include clustering, classification, prediction, and link analysis (associations). One of the most important data mining applications is that of mining association rules. Association rules, first introduced in 1993 [Agrawal1993], are used to identify relationships among a set of items in a database. These relationships are not based on inherent properties of the data themselves (as with functional dependencies), but rather based on co-occurrence of the data items. Example 1 illustrates association rules and their use.

Example 1: A grocery store has weekly specials for which advertising supplements are created for the local newspaper. When an item, such as peanut butter, has been designated to go on sale, management determines what other items are frequently purchased with peanut butter. They find that bread is purchased with peanut butter 30% of the time and that jelly is purchased with it 40% of the time. Based on these associations, special displays of jelly and bread are placed near the peanut butter which is on sale. They also decide not to put these items on sale. These actions are aimed at increasing overall sales volume by taking advantage of the frequency with which these items are purchased together.

There are two association rules mentioned in Example 1. The first one states that when peanut butter is purchased, bread is purchased 30% of the time. The second one states that 40% of the time when peanut butter is purchased so is jelly. Association rules are often used by retail stores to analyze market basket transactions. The discovered association rules can be used by management to increase the effectiveness (and reduce the cost) associated with advertising,

marketing, inventory, and stock location on the floor. Association rules are also used for other applications such as prediction of failure in telecommunications networks by identifying what events occur before a failure. Most of our emphasis in this paper will be on basket market analysis, however in later sections we will look at other applications as well.

The objective of this paper is to provide a thorough survey of previous research on association rules. In the next section we give a formal definition of association rules. Section 3 contains the description of sequential and parallel algorithms as well as other algorithms to find association rules. Section 4 provides a new classification and comparison of the basic algorithms. Section 5 presents generalization and extension of association rules. In Section 6 we examine the generation of association rules when the database is being modified. In appendices we provide information on different association rule products, data source and source code available in the market, and include a table summarizing notation used throughout the paper.

2 ASSOCIATION RULE PROBLEM

A formal statement of the association rule problem is [Agrawal1993] [Cheung1996c]:

Definition 1: Let $I = \{I_1, I_2, \ldots, I_m\}$ be a set of m distinct attributes, also called *literals*. Let D be a database, where each record (tuple) T has a unique identifier, and contains a set of items such that $T \subseteq I$ An *association rule* is an implication of the form $X \Rightarrow Y$, where $X, Y \subset I$, are sets of items called *itemsets*, and $X \cap Y = \phi$. Here, X is called antecedent, and Y consequent.

Two important measures for association rules, support (s) and confidence (α), can be defined as follows.

Definition 2: The *support (s)* of an association rule is the ratio (in percent) of the records that contain $X \cup Y$ to the total number of records in the database.

Therefore, if we say that the support of a rule is 5% then it means that 5% of the total records contain $X \cup Y$. Support is the statistical significance of an association rule. Grocery store managers probably would not be concerned about how peanut butter and bread are related if less than 5% of store transactions have this combination of purchases. While a high support is often desirable for association rules, this is not always the case. For example, if we were using association rules to predict the failure of telecommunications switching nodes based on what set

of events occur prior to failure, even if these events do not occur very frequently association rules showing this relationship would still be important.

Definition 3: For a given number of records, *confidence (α)* is the ratio (in percent) of the number of records that contain X ⋃ Y to the number of records that contain X.

Thus, if we say that a rule has a confidence of 85%, it means that 85% of the records containing X also contain Y. The confidence of a rule indicates the degree of correlation in the dataset between X and Y. Confidence is a measure of a rule's strength. Often a large confidence is required for association rules. If a set of events occur a small percentage of the time before a switch failure or if a product is purchased only very rarely with peanut butter, these relationships may not be of much use for management.

Mining of association rules from a database consists of finding all rules that meet the user-specified threshold support and confidence. The problem of mining association rules can be decomposed into two subproblems [Agrawal1994] as stated in Algorithm 1.

Algorithm 1. Basic:
Input:
I, D, s, α
Output:
Association rules satisfying s and α
Algorithm:
1) Find all sets of items which occur with a frequency that is greater than or equal to the user-specified threshold support, s.
2) Generate the desired rules using the large itemsets, which have user-specified threshold confidence, α.

The first step in Algorithm 1 finds *large* or *frequent itemsets.* Itemsets other than those are referred as *small itemsets*. Here an itemset is a subset of the total set of items of interest from the database. An interesting (and useful) observation about large itemsets is that:

If an itemset X is small, any superset of X is also small.

Of course the contrapositive of this statement (If X is a large itemset than so is any subset of X) is also important to remember. In the remainder of this paper we use L to designate the set of large itemsets. The second step in Algorithm 1 finds association rules using large itemsets

obtained in the first step. Example 2 illustrates this basic process for finding association rules from large itemsets.

Example 2: Consider a small database with four items I={Bread, Butter, Eggs, Milk} and four transactions as shown in Table 1. Table 2 shows all itemsets for I. Suppose that the minimum support and minimum confidence of an association rule are 40% and 60%, respectively. There are several potential association rules. For discussion purposes we only look at those in Table 3. At first, we have to find out whether all sets of items in those rules are large. Secondly, we have to verify whether a rule has a confidence of at least 60%. If the above conditions are satisfied for a rule, we can say that there is enough evidence to conclude that the rule holds with a confidence of 60%. Itemsets associated with the aforementioned rules are: {Bread, Butter}, and {Butter, Eggs}. The support of each individual itemset is at least 40% (see Table 2). Therefore, all of these itemsets are large. The confidence of each rule is presented in Table 3. It is evident that the first rule (Bread $\Rightarrow$ Butter) holds. However, the second rule (Butter $\Rightarrow$ Eggs) does not hold because its confidence is less than 60%.

Table 1 Transaction Database for Example 2

Transaction ID	Items
i	Bread, Butter, Eggs
T2	Butter, Eggs, Milk
T3	Butter
T4	Bread, Butter

Table 2 Support for Itemsets in Table 1 and Large Itemsets with a support of 40%

Itemset	Support, s	Large/Small
Bread	50%	Large
Butter	100%	Large
Eggs	50%	Large
Milk	25%	Small
Bread, Butter	50%	Large
Bread, Eggs	25%	Small
Bread, Milk	0%	Small
Butter, Eggs	50%	Large
Butter, Milk	25%	Small
Eggs, Milk	25%	Small
Bread, Butter, Eggs	25%	Small
Bread, Butter, Milk	0%	Small
Bread, Eggs, Milk	0%	Small
Butter, Eggs, Milk	25%	Small
Bread, Butter Eggs, Milk	0%	Small

Table 3 Confidence of Some Association Rules for Example 1 where α=60%

Rule	Confidence	Rule Hold
Bread $\Rightarrow$ Butter	100%	Yes
Butter $\Rightarrow$ Bread	50%	No
Butter $\Rightarrow$ Eggs	50%	No
Eggs $\Rightarrow$ Butter	100%	Yes

The identification of the large itemsets is computationally expensive [Agrawal1994]. However, once all sets of large itemsets ($l \in$ L) are obtained, there is a straightforward algorithm for finding association rules given in [Agrawal1994] which is restated in Algorithm 2.

Algorithm 2. Find Association Rules Given Large Itemsets:
Input:
I, D, s, α, L
Output:
Association rules satisfying s and α
Algorithm:
1) Find all nonempty subsets, *x*, of each large itemset, $l \in$ L
3) For every subset, obtain a rule of the form x$\Rightarrow$ (*l*-x) if the ratio of the frequency of occurrence of *l* to that of x is greater than or equal to the threshold confidence.

For example, suppose we want to see whether the first rule {Bread $\Rightarrow$ Butter) holds for Example 2. Here *l* = {Bread, Butter}, and x = {Bread}. Therefore, (*l*-x) = {Butter}. Now, the ratio of support(Bread, Butter) to support(Bread) is 100% which is greater than the minimum confidence. Therefore, the rule holds. For a better understanding, let us consider the third rule, Butter $\Rightarrow$ Eggs, where *x* = {Butter}, and (*l*-*x*) = {Eggs}. The ratio of support(Butter, Eggs) to support(Butter) is 50% which is less than 60%. Therefore, we can say that there is not enough evidence to conclude {Butter} $\Rightarrow$ {Eggs} with 60% confidence.

Since finding large itemsets in a huge database is very expensive and dominates the overall cost of mining association rules, most research has been focused on developing efficient algorithms to solve step 1 in Algorithm 1 [Agrawal1994] [Cheung1996c] [Klemettinen1994]. The following section provides an overview of these algorithms.

3 BASIC ALGORITHMS

In this section we provide a survey of existing algorithms to generate association rules. Most algorithms used to identify large itemsets can be classified as either sequential or parallel. In most cases, it is assumed that the itemsets are identified and stored in lexicographic order (based on item name). This ordering provides a logical manner in which itemsets can be generated and counted. This is the normal approach with sequential algorithms. On the other hand, parallel algorithms focus on how to parallelize the task of finding large itemsets. In the following subsections we describe important features of previously proposed algorithms. We describe all techniques, but only include a statement of the algorithm and survey its use with an example for a representative subset of these algorithms. We discuss the performance of the algorithms and look at data structures used.

3.1 Sequential Algorithms

3.1.1 AIS

The AIS algorithm was the first published algorithm developed to generate all large itemsets in a transaction database [Agrawal1993]. It focused on the enhancement of databases with necessary functionality to process decision support queries. This algorithm was targeted to discover qualitative rules. This technique is limited to only one item in the consequent. That is, the association rules are in the form of $X \Rightarrow I_j \mid \alpha$, where X is a set of items and I_j is a single item in the domain I, and α is the confidence of the rule.

The AIS algorithm makes multiple passes over the entire database. During each pass, it scans all transactions. In the first pass, it counts the support of individual items and determines which of them are large or frequent in the database. Large itemsets of each pass are extended to generate candidate itemsets. After scanning a transaction, the common itemsets between large itemsets of the previous pass and items of this transaction are determined. These common itemsets are extended with other items in the transaction to generate new candidate itemsets. A large itemset *l* is extended with only those items in the transaction that are large and occur in the lexicographic ordering of items later than any of the items in *l*. To perform this task efficiently, it uses an estimation tool and pruning technique. The estimation and pruning techniques determine

candidate sets by omitting unnecessary itemsets from the candidate sets. Then, the support of each candidate set is computed. Candidate sets having supports greater than or equal to min support are chosen as large itemsets. These large itemsets are extended to generate candidate sets for the next pass. This process terminates when no more large itemsets are found.

It is believed that if an itemset is absent in the whole database, it can never become a candidate for measurement of large itemsets in the subsequent pass. To avoid replication of an itemset, items are kept in lexicographic order. An itemset A is tried for extension only by items B (i.e., B=I_1, I_2, ……I_k) that are later in the ordering than any of the members of A. For example, let I={p, q, r, s, t, u, v}, and {p, q} be a large itemset. For transaction T = {p, q, r, s}, the following candidate itemsets are generated:

{p, q, r} expected large: continue extending

{p, q, s} expected large: cannot be extended further

{p, q, r, s} expected small: cannot be extended further

Let us see how the expected support for A+B is calculated. The expected support of A+B is the product of individual relative frequencies of items in B and the support for A, which is given as follows [Agrawal1993]:

$$s_{expected} = f(I_1) \times f(I_2) \times \ldots \times f(I_k) \times (x\text{-}c)/dbsize$$

where $f(I_i)$ represents the relative frequency of item I_i in the database, and (x-c)/dbsize is the actual support for A in the remaining portion of the database (here x = number of transactions that contain itemset A, c = number of transactions containing A that have already been processed in the current pass, and dbsize = the total number of transactions in the database).

Generation of a huge number of candidate sets might cause the memory buffer to overflow. Therefore, a suitable buffer management scheme is required to handle this problem whenever necessary. The AIS algorithm suggested that the large itemsets need not be in memory during a pass over the database and can be disk-resident. The memory buffer management algorithm for candidate sets is given in [Agrawal1993]. Two candidate itemsets U and V are called siblings if they are 1-extension (i.e. extension of an itemset with 1 item) of the same itemset. At first, an attempt is made to make room for new itemsets that have never been extended. If this attempt fails, the candidate itemset having the maximum number of items is discarded. All of its siblings are also discarded because their parents will have to be included in

the candidate itemsets for the next pass. Even after pruning, there might be a situation that all the itemsets that need to be measured in a pass may not fit into memory.

Applying to sales data obtained from a large retailing company, the effectiveness of the AIS algorithm was measured in [Agrawal1993]. There were a total of 46,873 customer transactions and 63 departments in the database. The algorithm was used to find if there was an association between departments in the customers' purchasing behavior. The main problem of the AIS algorithm is that it generates too many candidates that later turn out to be small [Agrawal1994]**.** Besides the single consequent in the rule, another drawback of the AIS algorithm is that the data structures required for maintaining large and candidate itemsets were not specified [Agrawal1993]**.** If there is a situation where a database has m items and all items appear in every transaction, there will be 2^m potentially large itemsets. Therefore, this method exhibits complexity which is exponential in the order of m in the worst case.

3.1.2 SETM

The SETM algorithm was proposed in [Houtsma1995] and was motivated by the desire to use SQL to calculate large itemsets [Srikant1996b]. In this algorithm each member of the set large itemsets, $\overline{L_k}$, is in the form <TID, itemset> where TID is the unique identifier of a transaction. Similarly, each member of the set of candidate itemsets, $\overline{C_k}$, is in the form <TID, itemset>.

Similar to the AIS algorithm, the SETM algorithm makes multiple passes over the database. In the first pass, it counts the support of individual items and determines which of them are large or frequent in the database. Then, it generates the candidate itemsets by extending large itemsets of the previous pass. In addition, the SETM remembers the TIDs of the generating transactions with the candidate itemsets. The relational merge-join operation can be used to generate candidate itemsets [Srikant1996b]. Generating candidate sets, the SETM algorithm saves a copy of the candidate itemsets together with TID of the generating transaction in a sequential manner. Afterwards, the candidate itemsets are sorted on itemsets, and small itemsets are deleted by using an aggregation function. If the database is in sorted order on the basis of TID, large itemsets contained in a transaction in the next pass are obtained by sorting $\overline{L_k}$ on TID.

This way, several passes are made on the database. When no more large itemsets are found, the algorithm terminates.

The main disadvantage of this algorithm is due to the number of candidate sets $\overline{C_k}$ [Agrawal1994]. Since for each candidate itemset there is a TID associated with it, it requires more space to store a large number of TIDs. Furthermore, when the support of a candidate itemset is counted at the end of the pass, $\overline{C_k}$ is not in ordered fashion. Therefore, again sorting is needed on itemsets. Then, the candidate itemsets are pruned by discarding the candidate itemsets which do not satisfy the support constraint. Another sort on TID is necessary for the resulting set ($\overline{L_k}$). Afterwards, $\overline{L_k}$ can be used for generating candidate itemsets in the next pass. No buffer management technique was considered in the SETM algorithm [Agrawal1994]. It is assumed that $\overline{C_k}$ can fit in the main memory. Furthermore, [Sarawagi1998] mentioned that SETM is not efficient and there are no results reported on running it against a relational DBMS.

3.1.3 Apriori

The Apriori algorithm developed by [Agrawal1994] is a great achievement in the history of mining association rules [Cheung1996c]. It is by far the most well-known association rule algorithm. This technique uses the property that any subset of a large itemset must be a large itemset. Also, it is assumed that items within an itemset are kept in lexicographic order. The fundamental differences of this algorithm from the AIS and SETM algorithms are the way of generating candidate itemsets and the selection of candidate itemsets for counting. As mentioned earlier, in both the AIS and SETM algorithms, the common itemsets between large itemsets of the previous pass and items of a transaction are obtained. These common itemsets are extended with other individual items in the transaction to generate candidate itemsets. However, those individual items may not be large. As we know that a superset of one large itemset and a small itemset will result in a small itemset, these techniques generate too many candidate itemsets which turn out to be small. The Apriori algorithm addresses this important issue. The Apriori generates the candidate itemsets by joining the large itemsets of the previous pass and deleting those subsets which are small in the previous pass without considering the transactions in the database. By only considering large itemsets of the previous pass, the number of candidate large itemsets is significantly reduced.

In the first pass, the itemsets with only one item are counted. The discovered large itemsets of the first pass are used to generate the candidate sets of the second pass using the apriori_gen() function. Once the candidate itemsets are found, their supports are counted to discover the large itemsets of size two by scanning the database. In the third pass, the large itemsets of the second pass are considered as the candidate sets to discover large itemsets of this pass. This iterative process terminates when no new large itemsets are found. Each pass *i* of the algorithm scans the database once and determines large itemsets of size *i*. L_i denotes large itemsets of size *i*, while C_i is candidates of size *i*.

The apriori_gen() function as described in [Agrawal1994] has two steps. During the first step, L_{k-1} is joined with itself to obtain C_k. In the second step, apriori_gen() deletes all itemsets from the join result, which have some (k-1)–subset that is not in L_{k-1}. Then, it returns the remaining large k-itemsets.

Method: apriori_gen() [Agrawal1994]
Input: set of all large (k-1)-itemsets L_{k-1}
Output: A superset of the set of all large k-itemsets
//Join step
I_i = Items i
insert into C_k
    Select $p.I_1$, $p.I_2$, ……., $p.I_{k-1}$, $q.I_{k-1}$
    From L_{k-1} is p, L_{k-1} is q
    Where $p.I_1 = q.I_1$ and …… and $p.I_{k-2} = q.I_{k-2}$ and $p.I_{k-1} < q.I_{k-1}$.
//pruning step
forall itemsets $c \in C_k$ do
    forall (k-1)-subsets s of c do
        If ($s \notin L_{k-1}$) then
            delete c from C_k

Consider the example given in Table 4 to illustrate the apriori_gen(). Large itemsets after the third pass are shown in the first column. Suppose a transaction contains {Apple, Bagel, Chicken, Eggs, DietCoke}. After joining L_3 with itself, C_4 will be {{Apple, Bagel, Chicken, DietCoke}, {Apple, Chicken, DietCoke, Eggs}. The prune step deletes the itemset {Apple, Chicken, DietCoke, Eggs} because its subset with 3 items {Apple, DietCoke, Eggs} is not in L_3.

Table 4 Finding Candidate Sets Using Apriori_gen()

Large Itemsets in the third pass (L_3)	Join (L_3, L_3)	Candidate sets of the fourth pass (C_4 after pruning)
{{Apple, Bagel, Chicken}, {Apple, Bagel, DietCoke}, {Apple, Chicken, DietCoke}, {Apple, Chicken, Eggs}, {Bagel, Chicken, DietCoke}}	{{Apple, Bagel, Chicken, DietCoke}, {Apple, Chicken, DietCoke Eggs}}	{{Apple, Bagel, Chicken, DietCoke}}

The subset() function returns subsets of candidate sets that appear in a transaction. Counting support of candidates is a time-consuming step in the algorithm [Cengiz1997]. To reduce the number of candidates that need to be checked for a given transaction, candidate itemsets C_k are stored in a hash tree. A node of the hash tree either contains a leaf node or a hash table (an internal node). The leaf nodes contain the candidate itemsets in sorted order. The internal nodes of the tree have hash tables that link to child nodes. Itemsets are inserted into the hash tree using a hash function. When an itemset is inserted, it is required to start from the root and go down the tree until a leaf is reached. Furthermore, L_k are stored in a hash table to make the pruning step faster [Srikant1996b]

Algorithm 3 shows the Apriori technique. As mentioned earlier, the algorithm proceeds iteratively.

Function count(C: a set of itemsets, D: database)
begin
 for each transaction T $\in$ D=$\bigcup D^i$ **do begin**
  forall subsets x $\subseteq$ T **do**
   if x $\in$ C **then**
    x.count++;
 end
end

Algorithm 3. Apriori [Agrawal1994]
Input:
I, D, s
Output:
L
Algorithm:
//Apriori Algorithm proposed by Agrawal R., Srikant, R. [**Agrawal1994**]
//procedure LargeItemsets
1) C_1: = I; //Candidate 1-itemsets
2) Generate L_1 by traversing database and counting each occurrence of an attribute in a transaction;
3) **for** ($k = 2$; $L_{k-1} \neq \phi$; k++) **do begin**
//Candidate Itemset generation
//New k-candidate itemsets are generated from (k-1)-large itemsets
4) C_k = apriori-gen(L_{k-1});
//Counting support of C_k
5) Count (C_k, D)
6) $L_k = \{c \in C_k \mid c.count \geq minsup\}$
7) **end**
9) $L := \bigcup_k L_k$

Figure 1 illustrates how the Apriori algorithm works on Example 2. Initially, each item of the itemset is considered as a 1-item candidate itemset. Therefore, C_1 has four 1-item candidate sets which are {Bread}, {Butter}, {Eggs}, and {Milk}. L_1 consists of those 1-itemsets from C_1 with support greater than or equal to 0.4. C_2 is formed by joining L_1 with itself, and deleting any itemsets which have subsets not in L_1. This way, we obtain C_2 as {{Bread Butter}, {Bread Eggs}, {Butter Eggs}}. Counting support of C_2, L_2 is found to be {{Bread Butter}, {Butter Eggs}}. Using apriori_gen(), we do not get any candidate itemsets for the third round. This is because the conditions for joining L_2 with itself are not satisfied.

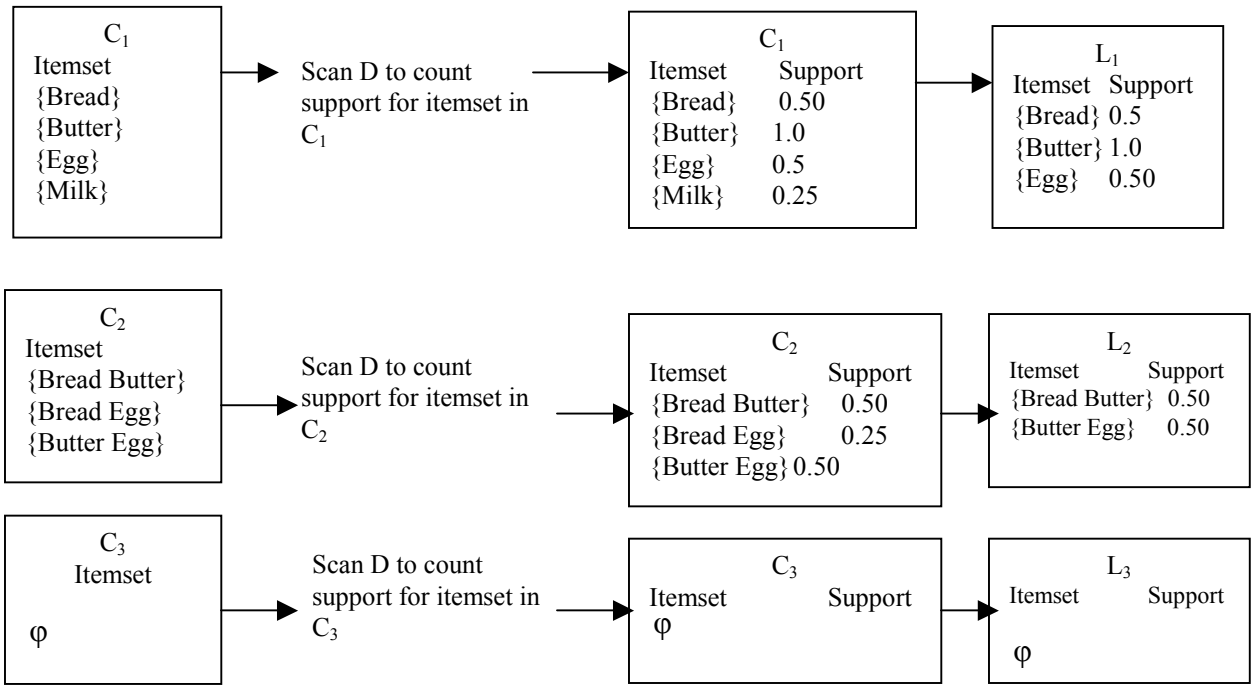

Figure 1 Discovering Large Itemsets using the Apriori Algorithm

Apriori always outperforms AIS and SETM (Agrawal1994). Recall the example given in Table 4. Apriori generates only one candidate ({Apple, Bagel, Chicken, DietCoke}) in the fourth round. On the other hand, AIS and SETM will generate five candidates which are {Apple, Bagel, Chicken, DietCoke}, {Apple, Bagel, Chicken, Eggs}, {Apple, Bagel, DietCoke, Eggs}, {Apple, Chicken, DietCoke, Eggs}, and {Bagel, Chicken, DietCoke, Eggs}. As we know, large itemsets are discovered after counting the supports of candidates. Therefore, unnecessary support counts are required if either AIS or SETM is followed.

Apriori incorporates buffer management to handle the fact that all the large itemsets L_{k-1} and the candidate itemsets C_k need to be stored in the candidate generation phase of a pass k may not fit in the memory. A similar problem may arise during the counting phase where storage for C_k and at least one page to buffer the database transactions are needed [Agrawal1994]. [Agrawal1994] considered two approaches to handle these issues. At first they assumed that L_{k-1} fits in memory but C_k does not. The authors resolve this problem by modifying apriori_gen() so that it generates a number of candidate sets C_k' which fits in the memory. Large itemsets L_k resulting from C_k' are written to disk, while small itemsets are deleted. This process continues

until all of C_k has been measured. The second scenario is that L_{k-1} does not fit in the memory. This problem is handled by sorting L_{k-1} externally [Srikant1996b]. A block of L_{k-1} is brought into the memory in which the first (k-2) items are the same. Blocks of L_{k-1} are read and candidates are generated until the memory fills up. This process continues until all C_k has been counted.

The performance of Apriori was assessed by conducting several experiments for discovering large itemsets on an IBM RS/6000 530 H workstation with the CPU clock rate of 33 MHz, 64 MB of main memory, and running AIX 3.2. Experimental results show that the Apriori algorithm always outperforms both AIS and SETM [Agrawal1994].

3.1.4 Apriori-TID

As mentioned earlier, Apriori scans the entire database in each pass to count support. Scanning of the entire database may not be needed in all passes. Based on this conjecture**,** [Agrawal1994] proposed another algorithm called Apriori-TID. Similar to Apriori, Apriori-TID uses the Apriori's candidate generating function to determine candidate itemsets before the beginning of a pass. The main difference from Apriori is that it does not use the database for counting support after the first pass. Rather, it uses an encoding of the candidate itemsets used in the previous pass denoted by $\overline{C_k}$. As with SETM, each member of the set $\overline{C_k}$ is of the form <TID, X_k> where X_k is a potentially large k-itemset present in the transaction with the identifier TID. In the first pass, $\overline{C_1}$ corresponds to the database. However, each item is replaced by the itemset. In other passes, the member of $\overline{C_k}$ corresponding to transaction T is <TID, c> where c is a candidate belonging to C_k contained in T. Therefore, the size of $\overline{C_k}$ may be smaller than the number of transactions in the database. Furthermore, each entry in $\overline{C_k}$ may be smaller than the corresponding transaction for larger k values. This is because very few candidates may be contained in the transaction. It should be mentioned that each entry in $\overline{C_k}$ may be larger than the corresponding transaction for smaller k values [Srikant1996b].

At first, the entire database is scanned and $\overline{C_1}$ is obtained in terms of itemsets. That is, each entry of $\overline{C_1}$ has all items along with TID. Large itemsets with 1-item L_1 are calculated by counting entries of $\overline{C_1}$. Then, apriori_gen() is used to obtain C_2. Entries of $\overline{C_2}$ corresponding to

a transaction T is obtained by considering members of C_2 which are present in T. To perform this task, $\overline{C_1}$ is scanned rather than the entire database. Afterwards, L_2 is obtained by counting the support in $\overline{C_2}$. This process continues until the candidate itemsets are found to be empty.

The advantage of using this encoding function is that in later passes the size of the encoding function becomes smaller than the database, thus saving much reading effort. Apriori-TID also outperforms AIS and SETM. Using the example given in Table 4, where L_3 was found as {{Apple, Bagel, Chicken}, {Apple, Bagel, DietCoke}, {Apple, Chicken, DietCoke}, {Apple, Chicken, Eggs}, {Bagel, Chicken, DietCoke}}. Similar to Apriori, Apriori-TID will also generate only one candidate itemsets {Apple, Bagel, Chicken, DietCoke}. As mentioned earlier, both AIS and SETM generate five candidate itemsets which are {Apple, Bagel, Chicken, DietCoke}, {Apple, Bagel, Chicken, Eggs}, {Apple, Bagel, DietCoke Eggs}, {Apple, Chicken, DietCoke, Eggs}, and {Bagel, Chicken, DietCoke, Eggs}.

In Apriori-TID, the candidate itemsets in C_k are stored in an array indexed by TIDs of the itemsets in C_k. Each C_k is stored in a sequential structure. In the k^{th} pass, Apriori-TID needs memory space for L_{k-1} and C_k during candidate generation. Memory space is needed for C_{k-1}, C_k, $\overline{C_k}$, and $\overline{C_{k-1}}$ in the counting phase. Roughly half of the buffer is filled with candidates at the time of candidate generation. This allows the relevant portions of both C_k and C_{k-1} to be kept in memory during the computing phase. If L_k does not fit in the memory, it is recommended to sort L_k externally.

Similar to Apriori, the performance of this algorithm was also assessed by experimenting using a large sample on an IBM RS/6000 530H workstation [Agrawal1994]. Since Apriori-TID uses $\overline{C_k}$ rather than the entire database after the first pass, it is very effective in later passes when $\overline{C_k}$ becomes smaller. However, Apriori-TID has the same problem as SETM in that $\overline{C_k}$ tends to be large, but Apriori-TID generates significantly fewer candidate itemsets than SETM does. Apriori-TID does not need to sort $\overline{C_k}$ as is needed in SETM. There is some problem associated with buffer management when $\overline{C_k}$ becomes larger. It was also found that Apriori-TID outperforms Apriori when there is a smaller number of $\overline{C_k}$ sets, which can fit in the memory and the distribution of the large itemsets has a long tail [Srikant1996b]. That means the distribution of entries in large itemsets is high at early stage. The distribution becomes smaller immediately

after it reaches the peak and continues for a long time. It is reported that the performance of Apriori is better than that of Apriori-TID for large data sets [Agrawal1994]. On the other hand, Apriori-TID outperforms Apriori when the $\overline{C_k}$ sets are relatively small (fit in memory). Therefore, a hybrid technique "Apriori-Hybrid" was also introduced by [Agrawal1994].

3.1.5 Apriori-Hybrid

This algorithm is based on the idea that it is not necessary to use the same algorithm in all passes over data. As mentioned in [Agrawal1994], Apriori has better performance in earlier passes, and Apiori-TID outperforms Apriori in later passes. Based on the experimental observations, the Apriori-Hybrid technique was developed which uses Apriori in the initial passes and switches to Apriori-TID when it expects that the set $\overline{C_k}$ at the end of the pass will fit in memory. Therefore, an estimation of $\overline{C_k}$ at the end of each pass is necessary. Also, there is a cost involvement of switching from Apriori to Apriori-TID. The performance of this technique was also evaluated by conducting experiments for large datasets. It was observed that Apriori-Hybrid performs better than Apriori except in the case when the switching occurs at the very end of the passes [Srikant1996b].

3.1.6 Off-line Candidate Determination (OCD)

The Off-line Candidate Determination (OCD) technique is proposed in [Mannila1994] based on the idea that small samples are usually quite good for finding large itemsets. The OCD technique uses the results of the combinatorial analysis of the information obtained from previous passes to eliminate unnecessary candidate sets. To know if a subset $Y \subseteq I$ is infrequent, at least (1-s) of the transactions must be scanned where s is the support threshold. Therefore, for small values of s, almost the entire relation has to be read. It is obvious that if the database is very large, it is important to make as few passes over the data as possible.

OCD follows a different approach from AIS to determine candidate sets. OCD uses all available information from previous passes to prune candidate sets between the passes by keeping the pass as simple as possible. It produces a set L_k as the collection of all large itemsets of size k. Candidate sets C_{k+1} contain those sets of size (k+1) that can possibly be in L_{k+1}, given

the large itemsets of L_k. It is noted that if $X \in L_{k+e}$ and $e \geq 0$, then X includes $\binom{k+e}{k}$ sets from L_k where e denotes the extension of L_k. That means, if e=1, k=2, and $X \in L_3$, then X includes $\binom{3}{2}$ or 3 sets from L_2. Similarly, each item of L_4 includes 4 items of L_3, and so on. For example, we know L_2={{Apple, Banana}, {Banana, Cabbage}, {Apple, Cabbage}, {Apple, Eggs}, {Banana, Eggs}, {Apple, Icecream}, {Cabbage, Syrup}}. We can conclude that {Apple, Banana, Cabbage} and {Apple, Banana, Eggs} are the only possible members of L_3. This is because they are the only sets of size 3 whose all subsets of size 2 are included in L_2. At this stage, L_4 is empty. This is because any member of L_4 includes 4 items of L_3, but we have only 2 members in L_3. Therefore, C_{k+1} is counted as follows:

$$C_{k+1} = \{Y \subseteq I \text{ such that } |Y|=k+1 \text{ and } Y \text{ includes } (k+1) \text{ members of } L_k\} \quad (1)$$

A trivial solution for finding C_{k+1} is the exhaustive procedure. In the exhaustive method, all subsets of size k+1 are inspected. However, this procedure produces a large number of unnecessary candidates, and it is a wasteful technique. To expedite the counting operation, OCD suggests two alternative approaches. One of them is to compute a collection of C'_{k+1} by forming unions of L_k that have (k-1) items in common as mentioned Equation (2):

$$C'_{k+1} = \{Y \bigcup Y' \text{ such that } Y, Y' \in L_k \text{ and } |Y \bigcup Y'| = (k-1)\} \quad (2)$$

Then $C_{k+1} \in C'_{k+1}$ and C_{k+1} can be computed by checking for each set in C'_{k+1} whether the defining condition of C_{k+1} holds.

The second approach is to form unions of sets from L_k and L_1 as expressed in Equation (3):

$$C''_{k+1} = \{Y \bigcup Y' \text{ such that } Y \in L_k \text{ and } Y' \in L_1 \text{ and } Y' \not\subset Y\} \quad (3)$$

Then compute C_{k+1} by checking the inclusion condition as stated in Equation (1).

Here it is noted that the work involved in generating C_{k+1} does not depend on the size of database, rather on the size of L_k. Also, one can compute several families of $C_{k+1}, C_{k+2}, \ldots, C_{k+e}$ for some e>1 directly from L_k. The time complexity for determining C_{k+1} from C'_{k+1} is $O(k|L_k|^3)$. On the other hand, the running time for determining C_{k+1} from C''_{k+1} is linear in size of the database (n) and exponential in size of the largest large itemset. Therefore, the algorithm can be

quite slow for very large values of n. A good approximation of the large itemsets can be obtained by analyzing only small samples of a large database [Lee1998; Mannila1994]. Theoretical analysis performed by [Mannila1994] shows that small samples are quite good for finding large itemsets. It is also mentioned in [Mannila1994] that even for fairly low values of support threshold, a sample consisting of 3000 rows gives an extremely good approximation in finding large itemsets.

The performance of this algorithm was evaluated in [Mannila1994] by using two datasets. One of them is a course enrollment database of 4734 students. The second one is a telephone company fault management database which contains some 30,000 records of switching network notifications. Experimental results indicate that the time requirement of OCD is typically 10-20% of that of AIS. The advantage of OCD increases with a lower support threshold [Mannila1994]. Generated candidates in AIS are significantly higher than those in OCD. AIS may generate duplicate candidates during the pass whereas OCD generates any candidate once and checks that its subsets are large before evaluating it against the database.

3.1.7 Partitioning

PARTITION [Savasere1995] reduces the number of database scans to 2. It divides the database into small partitions such that each partition can be handled in the main memory. Let the partitions of the database be D^1, D^2, ..., D^p. In the first scan, it finds the *local large itemsets* in each partition D^i ($1 \le i \le p$), i.e. $\{X \mid X.count \ge s \times |D^i|\}$. The local large itemsets, L^i, can be found by using a level-wise algorithm such as Apriori. Since each partition can fit in the main memory, there will be no additional disk I/O for each partition after loading the partition into the main memory. In the second scan, it uses the property that a large itemset in the whole database must be locally large in at least one partition of the database. Then the union of the local large itemsets found in each partition are used as the candidates and are counted through the whole database to find all the large itemsets.

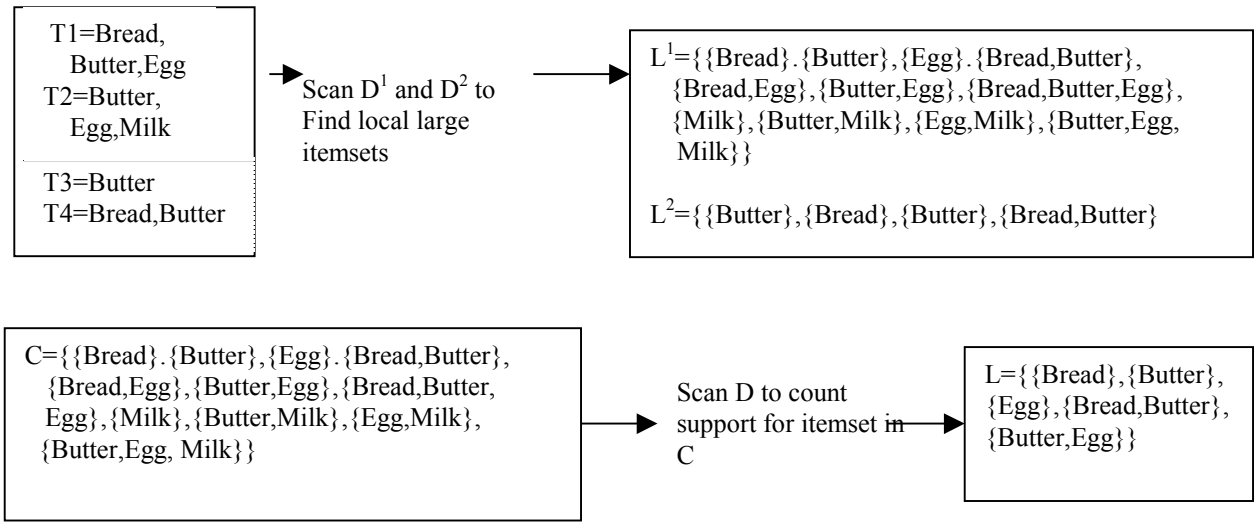

Figure 2 Discovering Large Itemsets using the PARTITION Algorithm

Figure 2 illustrates the use of PARTITION with Example 2. If the database is divided into two partitions, with the first partition containing the first two transactions and the second partition the remaining two transactions. Since the minimum support is 40% and there are only two transactions in each partition, an itemset which occurs once will be large. Then the local large itemsets in the two partitions are just all subsets of the transactions. Their union is the set of the candidate itemsets for the second scan. The algorithm is shown in Algorithm 4. Note that we use superscripts to denote the database partitions, and subscripts the sizes of the itemsets.

Algorithm 4. PARTITION [Savasere 95]
Input:
I, s, D^1, D^2, ..., D^p
Output:
L
Algorithm:
//scan one computes the local large itemsets in each partition
1) **for** i from 1 to p **do**
2) L^i = Apriori(I,D^i,s); //L^i are all local large itemsets(all sizes) in D^i
//scan two counts the union of the local large itemsets in all partitions
3) $C = \bigcup_i L^i$;
4) count(C, D) = $\bigcup D^i$;
5) return $L = \{x \mid x \in C, x.count \geq s \times |D|\}$;

PARTITION favors a homogeneous data distribution. That is, if the count of an itemset is evenly distributed in each partition, then most of the itemsets to be counted in the second scan will be large. However, for a skewed data distribution, most of the itemsets in the second scan may turn out to be small, thus wasting a lot of CPU time counting false itemsets. AS-CPA (Anti-Skew Counting Partition Algorithm) [Lin1998] is a family of anti-skew algorithms, which were proposed to improve PARTITION when data distribution is skewed. In the first scan, the counts of the itemsets found in the previous partitions will be accumulated and incremented in the later partitions. The accumulated counts are used to prune away the itemsets that are likely to be small. Due to the early pruning techniques, the number of false itemsets to be counted in the second scan is reduced.

3.1.8 Sampling

Sampling [Toivonen1996] reduces the number of database scans to one in the best case and two in the worst. A sample which can fit in the main memory is first drawn from the database. The set of large itemsets in the sample is then found from this sample by using a level-wise algorithm such as Apriori. Let the set of large itemsets in the sample be PL, which is used as a set of probable large itemsets and used to generate candidates which are to be verified against the whole database . The candidates are generated by applying the negative border function, BD^{-}, to PL. Thus the candidates are $BD^{-}(PL) \cup PL$. The negative border of a set of itemsets PL is the minimal set of itemsets which are not in PL, but all their subsets are. The negative border function is a generalization of the apriori_gen function in Apriori. When all itemsets in PL are of the same size, $BD^{-}(PL)$ = apriori_gen(PL). The difference lies in that the negative border can be applied to a set of itemsets of different sizes, while the function apriori_gen() only applies to a single size. After the candidates are generated, the whole database is scanned once to determine the counts of the candidates. If all large itemsets are in PL, i.e., no itemsets in $BD^{-}(PL)$ turn out to be large, then all large itemsets are found and the algorithm terminates. This can guarantee that all large itemsets are found, because $BD^{-}(PL) \cup PL$ actually contains all candidate itemsets of Apriori if PL contains all large itemsets L, i.e., $L \subseteq PL$. Otherwise, i.e. there are misses in $BD^{-}(PL)$, some new candidate itemsets must be counted to ensure that all large itemsets are

found, and thus one more scan is needed. In this case, i.e., $L \bigcap PL \neq \varnothing$, the candidate itemsets in the first scan may not contain all candidate itemsets of Apriori.

To illustrate Sampling, suppose PL={{A}, {B}, {C}, {A,B}}. The candidate itemsets for the first scan are $BD^{-}(PL) \bigcup PL$ = {{A, C}, {B,C}} $\bigcup$ {{A}, {B}, {C}, {A,B}} = {{A}, {B}, {C}, {A,B}, {A,C}, {B,C}}. If L ={{A}, {B}, {C}, {A,B}, {A,C}, {B,C}}, i.e., there are two misses {A,C} and {B,C} in $BD^{-}(PL)$, the itemset {A, B, C}, which might be large, is a candidate in Apriori, while not counted in the first scan of Sampling. So the Sampling algorithm needs one more scan to count the new candidate itemsets like {A, B, C}. The new candidate itemsets are generated by applying the negative border function recursively to the misses. The algorithm is shown in Algorithm 5.

Algorithm 5. Sampling [Toivonen 96]
Input:
I, s, D
Output:
L
Algorithm:
//draw a sample and find the local large itemsets in the sample
1) Ds = a random sample drawn from D;
2) PL = Apriori(I,Ds,s);
//first scan counts the candidates generated from PL
3) $C = PL \bigcup BD^{-}(PL)$;
4) count(C, D);
//second scan counts additional candidates if there are misses in $BD^{-}(PL)$
5) $ML = \{x \mid x \in BD^{-}(PL), x.count \geq s \times |D|\}$; //ML are the misses
6) **if** $ML \neq \varnothing$ **then** //MC are the new candidates generated from the misses
7) $MC = \{x \mid x \in C, x.count \geq s \times |D|\}$;
8) **repeat**
9) $MC = MC \bigcup BD^{-}(MC)$;
10) **until** MC doesn't grow;
11) MC = MC - C); //itemsets in C have already been counted in scan one
12) count(MC, D);
13) return $L = \{x \mid x \in C \bigcup MC, x.count \geq s \times |D|\}$;

3.1.9 Dynamic Itemset Counting [Brin1997a]

DIC (Dynamic Itemset Counting) [Brin1997a] tries to generate and count the itemsets earlier, thus reducing the number of database scans. The database is viewed as intervals of transactions,

and the intervals are scanned sequentially. While scanning the first interval, the 1-itemsets are generated and counted. At the end of the first interval, the 2-itemsets which are potentially large are generated. While scanning the second interval, all 1-itemsets and 2-itemsets generated are counted. At the end of the second interval, the 3-itemsets that are potentially large are generated, and are counted during scanning the third interval together with the 1-itemsets and 2-itemsets. In general, at the end of the k^{th} interval, the (k+1)-itemsets which are potentially large are generated and counted together with the previous itemsets in the later intervals. When reaching the end of the database, it rewinds the database to the beginning and counts the itemsets which are not fully counted. The actual number of database scans depends on the interval size. If the interval is small enough, all itemsets will be generated in the first scan and fully counted in the second scan. It also favors a homogeneous distribution as does the PARTITION.

3.1.10 CARMA

CARMA (Continuous Association Rule Mining Algorithm) [Hidb1999] brings the computation of large itemsets online. Being online, CARMA shows the current association rules to the user and allows the user to change the parameters, minimum support and minimum confidence, at any transaction during the first scan of the database. It needs at most 2 database scans. Similar to DIC, CARMA generates the itemsets in the first scan and finishes counting all the itemsets in the second scan. Different from DIC, CARMA generates the itemsets on the fly from the transactions. After reading each transaction, it first increments the counts of the itemsets which are subsets of the transaction. Then it generates new itemsets from the transaction, if all immediate subsets of the itemsets are currently potentially large with respect to the current minimum support and the part of the database that is read. For more accurate prediction of whether an itemset is potentially large, it calculates an upper bound for the count of the itemset, which is the sum of its current count and an estimate of the number of occurrences before the itemset is generated. The estimate of the number of occurrences (called maximum misses) is computed when the itemset is first generated.

3.2 Parallel and Distributed Algorithms

The current parallel and distributed algorithms are based on the serial algorithm Apriori. An excellent survey given in [Zaki1999] classifies the algorithms by load-balancing strategy, architecture and parallelism. Here we focus on the parallelism used: *data parallelism* and *task parallelism* [Chat1997]. The two paradigms differ in whether the candidate set is distributed across the processors or not. In the data parallelism paradigm, each node counts the same set of candidates. In the task parallelism paradigm, the candidate set is partitioned and distributed across the processors, and each node counts a different set of candidates. The database, however, may or may not be partitioned in either paradigm theoretically. In practice for more efficient I/O it is usually assumed the database is partitioned and distributed across the processors.

In the data parallelism paradigm, a representative algorithm is the count distribution algorithm in [Agrawal1996]. The candidates are duplicated on all processors, and the database is distributed across the processors. Each processor is responsible for computing the *local support counts* of all the candidates, which are the support counts in its database partition. All processors then compute the *global support counts* of the candidates, which are the total support counts of the candidates in the whole database, by exchanging the local support counts (Global Reduction). Subsequently, large itemsets are computed by each processor independently. The data parallelism paradigm is illustrated in Figure 3 using the data in Table 1. The four transactions are partitioned across the three processors with processor 3 having two transactions T3 and T4, processor 1 having transaction T1 and processor 2 having transaction T2. The three candidate itemsets in the second scan are duplicated on each processor. The local support counts are shown after scanning the local databases.

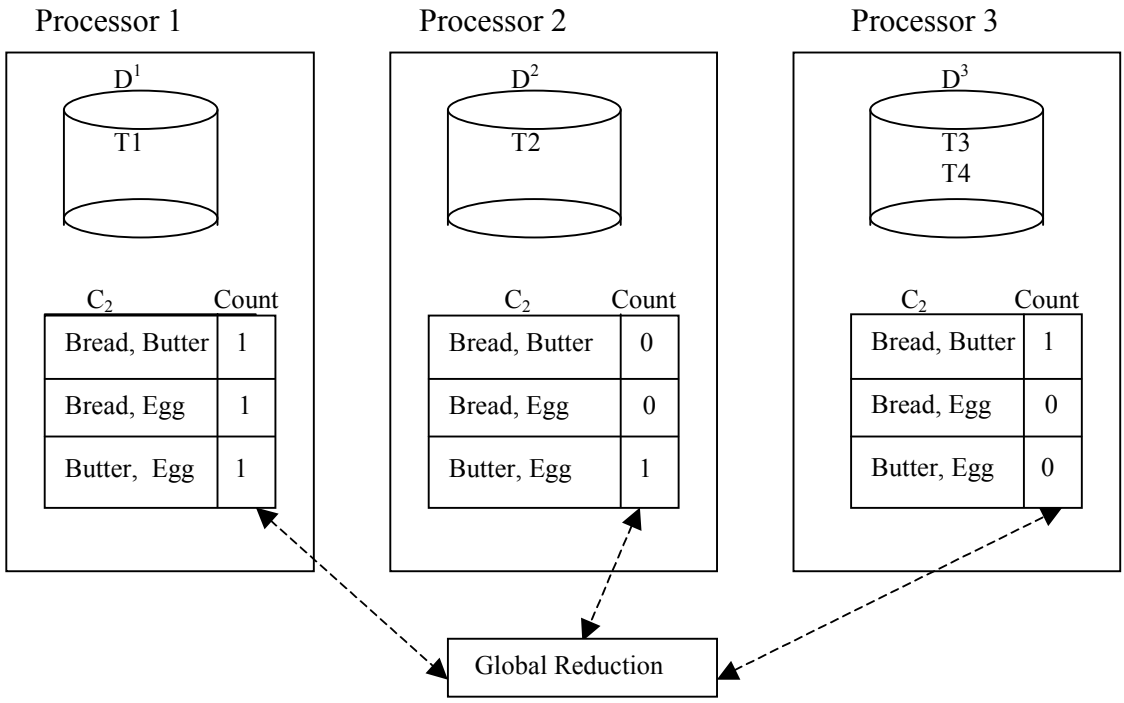

Figure 3 Data Parallelism Paradigm

In the task parallelism paradigm, a representative algorithm is the data distribution algorithm in [Agrawal1996]. The candidate set is partitioned and distributed across the processors as is the database. Each processor is responsible for keeping the global support counts of only a subset of the candidates. This approach requires two rounds of communication at each iteration. In the first round, every processor sends its database partition to all the other processors. In the second round, every processor broadcasts the large itemsets that it has found to all the other processors for computing the candidates for the next iteration. The task parallelism paradigm is shown in Figure 4 using the data in Table 1. The four transactions are partitioned as in data parallelism. The three candidate itemsets are partitioned across the processors with each processor having one candidate itemset. After scanning the local database and the database partitions broadcast from the other processors, the global count of each candidate is shown.

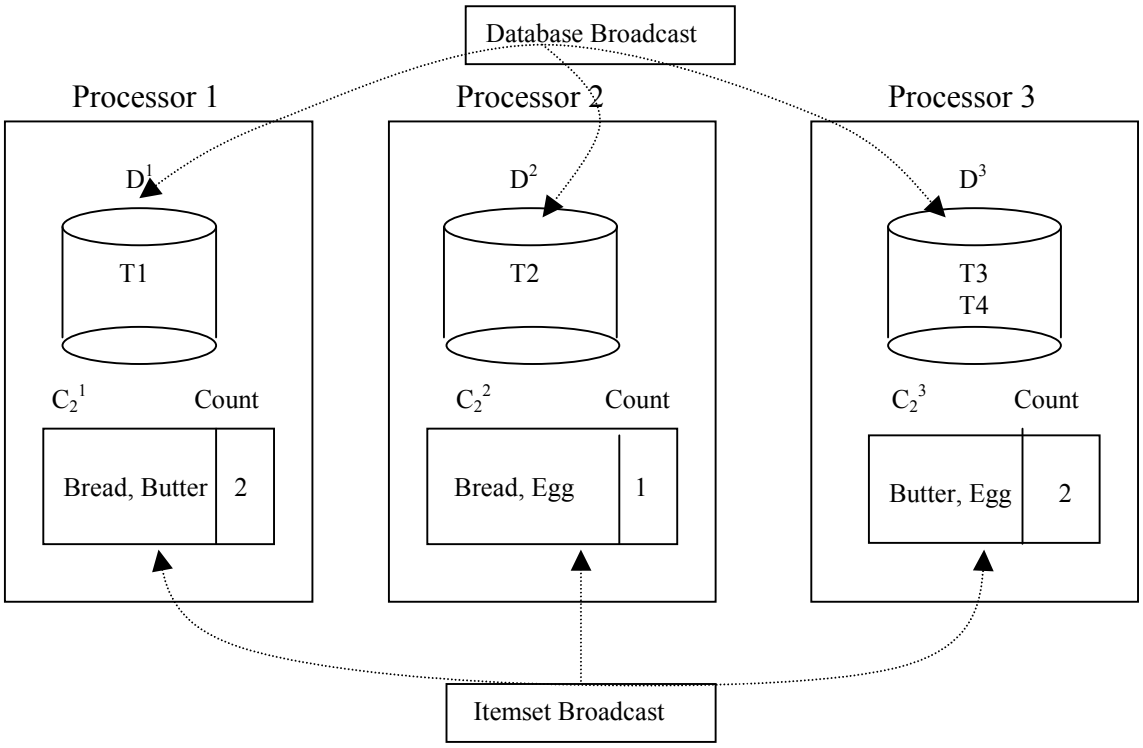

Figure 4 Task parallelism paradigm

3.2.1 Data Parallelism Algorithms

The algorithms which adopt the data parallelism paradigm include: CD [Agrawal1996], PDM [Park1995], DMA [Cheung1996], and CCPD [Zaki1996]. These parallel algorithms differ in whether further candidate pruning or efficient candidate counting techniques are employed or not. The representative algorithm CD(Count Distribution) is described in details, and for the other three algorithms only the additional techniques introduced are described.

3.2.1.1 CD

In CD, the database D is partitioned into $\{D^1, D^2, \ldots, D^p\}$ and distributed across n processors. Note that we use superscript to denote the processor number, while subscript the size of candidates. The program fragment of CD at processor i, $1 \le i \le p$, is outlined in Algorithm 6. There are basically three steps. In step 1, local support counts of the candidates C_k in the local database partition D^i are found. In step 2, each processor exchanges the local support counts of

all candidates to get the global support counts of all candidates. In step 3, the globally large itemsets L_k are identified and the candidates of size k+1 are generated by applying apriori_gen() to L_k on each processor independently. CD repeats steps 1 - 3 until no more candidates are found. CD was implemented on an IBM SP2 parallel computer, which is shared-nothing and communicates through the High-Performance Switch.

Algorithm 6 CD [Agrawal 1996]
Input:
I, s, D^1, D^2, …, D^p
Output:
L
Algorithm:
1) C_1=I;
2) **for** k=1;$C_k \neq \emptyset$;k++ **do begin**
//step one: counting to get the local counts
3) count(C_k, D^i); //local processor is *i*
//step two: exchanging the local counts with other processors
//to obtain the global counts in the whole database.
4) **forall** itemset $X \in C_k$ **do begin**
5) X.count=$\sum_{j=1}^{p}${X^j.count};
6) **end**
//step three: identifying the large itemsets and
//generating the candidates of size k+1
7) L_k={$c \in C_k$ | c.count $\geq s \times |D^1 \cup D^2 \cup \ldots \cup D^p|$};
8) C_{k+1}=apriori_gen(L_k);
9) **end**
10) return $L=L_1 \cup L_2 \cup \ldots \cup L_k$;

3.2.1.2 PDM

PDM (Parallel Data Mining) [Park1995a] is a modification of CD with inclusion of the direct hashing technique proposed in [Park1995]. The hash technique is used to prune some candidates in the next pass. It is especially useful for the second pass, as Apriori doesn't have any pruning in generating C_2 from L_1. In the first pass, in addition to counting all 1-itemsets, PDM maintains a hash table for storing the counts of the 2-itemsets. Note that in the hash table we don't need to store the 2-itemsets themselves but only the count for each bucket. For example, suppose {A, B} and {C} are large items and in the hash table for the 2-itemsets the bucket containing {AB, AD} turns out to be small (the count for this bucket is less than the minimum

support count). PDM will not generate AB as a size 2 candidate by the hash technique, while Apriori will generate AB as a candidate for the second pass, as no information about 2-itemsets can be obtained in the first pass. For the communication, in the k^{th} pass, PDM needs to exchange the local counts in the hash table for k+1-itemsets in addition to the local counts of the candidate k-itemsets.

3.2.1.3 DMA

DMA (Distributed Mining Algorithm) [Cheung1996] is also based on the data parallelism paradigm with the addition of candidate pruning techniques and communication message reduction techniques introduced. It uses the local counts of the large itemsets on each processor to decide whether a large itemset is *heavy* (both locally large in one database partition and globally large in the whole database), and then generates the candidates from the heavy large itemsets. For example, A and B are found heavy on processor 1 and 2 respectively, that is, A is globally large and locally large only on processor 1, B is globally large and locally large only on processor 2. DMA will not generate AB as a candidate 2-itemset, while Apriori will generate AB due to no consideration about the local counts on each processor. For the communication, instead of broadcasting the local counts of all candidates as in CD, DMA only sends the local counts to one polling site, thus reducing the message size from $O(p^2)$ to $O(p)$. DMA was implemented on a distributed network system initially, and was improved to a parallel version FPM(Fast Parallel Mining) on an IBM SP2 parallel machine [Cheung1998].

3.2.1.4 CCPD

CCPD (Common Candidate Partitioned Database) [Zaki1996] implements CD on a shared-memory SGI Power Challenge with some improvements. It proposes techniques for efficiently generating and counting the candidates in a shared-memory environment. It groups the large itemsets into equivalence classes based on the common prefixes (usually the first item) and generates the candidates from each equivalence class. Note that the grouping of the large itemsets will not reduce the number of candidates but reduce the time to generate the candidates. It also introduces a short-circuited subset checking method for efficient counting the candidates for each transaction.

3.2.2 Task Parallelism Algorithms

The algorithms adopting the task parallelism paradigm include: DD [Agrawal1996], IDD [Han1997], HPA [Shintani1996] and PAR [Zaki1997]. They all partition the candidates as well as the database among the processors. They differ in how the candidates and the database are partitioned. The representative algorithm DD (Data Distribution) [Agrawal1996] is described in more detail, and for the other algorithms only the different techniques are reviewed.

3.2.2.1 DD

In DD (Data Distribution) [Agrawal1996], the candidates are partitioned and distributed over all the processors in a round-robin fashion. There are three steps. In step one, each processor scans the local database partition to get the local counts of the candidates distributed to it. In step two, every processor broadcasts its database partition to the other processors and receives the other database partitions from the other processors, then scans the received database partitions to get global support counts in the whole database. In the last step, each processor computes the large itemsets in its candidate partition, exchanges with all others to get all the large itemsets, and then generates the candidates, partitions and distributes the candidates over all processors. These steps continue until there are no more candidates generated. Note that the communication overhead of broadcasting the database partitions can be reduced by asynchronous communication [Agrawal1996], which overlaps communication and computation. The details are described in Algorithm 7.

Algorithm 7. DD [Agrawal 1996]
Input:
$I,s,D^1, D^2, \ldots, D^p$
Output:
L
Algorithm:
1) $C_1^i \subseteq I$;
2) **for** ($k=1;C_k^i \neq \varnothing;k++$) **do begin**
//step one: counting to get the local counts
3) count(C_k^i , D^i); //local processor is *i*
//step two: broadcast the local database partition to others,
// receive the remote database partitions from others,

// scan D^j($1 \le j \le p$, $j \ne i$) to get the global counts.
4) broadcast(D^i);
5) **for** ($j=1$; ($j \le p$ and $j \ne i$);j++) **do begin**
6) receive(D^j) from processor j;
7) count(C_k^i , D^j);
8) **end**
//step three: identify the large itemsets in C^i_k,
// exchange with other processors to get all large itemsets C_k,
// generate the candidates of size k+1,
// partition the candidates and distribute over all processors.
9) $L_k^i = \{c | c \in C^i_k, c.count \ge s * |D^1 \cup D^2 \cup \ldots \cup D^p|\}$;
10) $L_k = \bigcup_{i=1}^{p} (L_k^i)$;
11) $C_{k+1} = apriori\_gen(L_k)$;
12) $C_{k+1}^i \subseteq C_{k+1}$; //partition the candidate itemsets across the processors
13) **end**
14) return $L = L_1 \cup L_2 \cup \ldots \cup L_k$;

3.2.2.2 IDD

IDD (Intelligent Data Distribution) is an improvement over DD [Han1997]. It partitions the candidates across the processors based on the first item of the candidates, that is, the candidates with the same first item will be partitioned into the same partition. Therefore, each processor needs to check only the subsets which begin with one of the items assigned to the processor. This reduces the redundant computation in DD, as for DD each processor needs to check all subsets of each transaction, which introduces a lot of redundant computation. To achieve a load-balanced distribution of the candidates, it uses a bin-packing technique to partition the candidates, that is, it first computes for each item the number of candidates that begin with the particular item, then it uses a bin-packing algorithm to assign the items to the candidate partitions such that the number of candidates in each partition is equal. It also adopts a ring architecture to reduce communication overhead, that is, it uses asynchronous point to point communication between neighbors in the ring instead of broadcasting.

3.2.2.3 HPA

HPA (Hash-based Parallel mining of Association rules) uses a hashing technique to distribute the candidates to different processors [Shintani1996], i.e., each processor uses the same hash function to compute the candidates distributed to it. In counting, it moves the subset

itemsets of the transactions to their destination processors by the same hash technique, instead of moving the database partitions among the processors. So one subset itemset of a transaction only goes to one processor instead of n. HPA was further improved by using the skew handling technique [Shintani1996]. The skew handling is to duplicate some candidates if there is available main memory in each processor, so that the workload of each processor is more balanced.

3.2.2.4 PAR

PAR (Parallel Association Rules) [Zaki1997] consists of a set of algorithms, which use different candidate partitioning and counting. They all assume a vertical database partition (tid lists for each item), contrast to the natural horizontal database partition (transaction lists). By using the vertical organization for the database, the counting of an itemset can simply be done by the intersection of the tid lists of the items in the itemset. However, they require a transformation to the vertical partition, if the database is horizontally organized. The database may be selectively duplicated to reduce synchronization. Two of the algorithms (Par-Eclat and Par-MaxEclat) use the equivalence class based on the first item of the candidates, while the other two (Par-Clique and Par-MaxClique) use the maximum hypergraph clique to partition the candidates. Note that in the hypergraph, a vertex is an item, an edge between k vertices corresponds to an itemset containing the items associated with the k vertices, and a clique is a sub-graph with all vertices in it connected. One feature of the algorithms(Par-MaxEclat and Par-MaxClique) is that it can find the maximal itemsets(the itemsets which are not any subset of the others). The itemset counting can be done bottom-up, top-down or hybrid. Since the algorithms need the large 2-itemsets to partition the candidates (either by equivalence class or by hypergraph clique), they use a preprocessing step to gather the occurrences of all 2-itemsets.

3.2.3 Other Parallel Algorithms

There are some other parallel algorithms which can not be classified into the two paradigms if strictly speaking. Although they share similar ideas with the two paradigms, they have distinct features. So we review these algorithms as other parallel algorithms. These parallel

algorithms include Candidate Distribution [Agrawal1996], SH(Skew Handling) [Harada1998] and HD(Hybrid Distribution) [Han1997].

3.2.3.1 Candidate Distribution

The candidate distributed algorithm [Agrawal1996] attempts to reduce the synchronization and communication overhead in the count distribution (CD) and data distribution (DD). In some pass *l*, which is heuristically determined, it divides the large itemsets L_{l-1} between the processors in such a way that a process can generate a unique set of candidates independent of the other processors. At the same time, the database is repartitioned so that a processor can count the candidates it generated independent of the others. Note that depending on the quality of the candidate partitioning, parts of the database may have to be replicated on several processors. The itemset partitioning is done by grouping the itemsets based on the common prefixes. After this candidate partition, each processor proceeds independently, counting only its portion of the candidates using only local database partition. No communication of counts or data tuples is ever required. Since before the candidate partition, it can use either the count distribution or the data distribution algorithm, the candidate distribution algorithm is a kind of hybrid of the two paradigms.

3.2.3.2 SH

In SH [Harada1998], the candidates are not generated a priori from the previous large itemsets, which seems different from the serial algorithm Apriori. Instead the candidates are generated independently by each processor on the fly while scanning the database partition. In iteration k, each processor generates and counts the k-itemsets from the transactions in its database partition. Only the k-itemsets all whose k k-1-subsets are globally large are generated, which is done by checking a bitmap for all the globally large k-1-itemsets. At the end of each iteration, all processors exchange the k-itemsets and their local counts, obtaining the global counts of all k-itemsets. The large k-itemsets are then identified and the bitmap for the large itemsets are also set on each processor. In case of workload imbalance in counting, the transactions are migrated from the busy processors to the idle processors. In case of insufficient

main memory, the current k-itemsets are sorted and spooled to the disk, and then the new k-itemsets are generated and counted for the rest of the database partition. At the end of the each iteration, the local counts of all k-itemsets are combined and exchanged with the other processors to get the global counts.

SH seems to be based on a different algorithm from Apriori, but it is very close to Apriori. First, it is iterative as Apriori, i.e., only at the end of an iteration are the new candidates of increased size generated. The difference from Apriori lies in when the candidates are generated, that is, SH generates the candidates from the transactions on the fly, while Apriori generates the candidates a priori at the end of each iteration. Second, the candidates generated by SH are exactly the same as those of Apriori if the database is evenly distributed. Only if the database is extremely skewed will the candidates be different. For example, if AB never occurs together(A and B can still be large items) in database partition *i*, i.e., its count is zero, SH will not generate AB as a candidate in the second pass on processor *i*. But if AB occurs together once, AB will be generated as a candidate by SH. Therefore, we can classify SH into the data parallelism paradigm with skew handling and insufficient main memory handling.

3.2.3.3 HD

HD (Hybrid Distribution) was proposed in [Han1997], which combines both paradigms. It assumes the p processors are arranged in a two dimensional grid of r rows and p/r columns. The database is partitioned equally among the p processors. The candidate set C_k is partitioned across the columns of this grid(i.e., p/n partitions with each column having one partition of candidate sets), and the partition of candidate sets on each column are duplicated on all processors along each row for that column. Now, any data distribution algorithm can be executed independently along each column of the grid, and the global counts of each subset of C_k are obtained by performing a reduction operation along each row of the grid as in the data parallelism paradigm. The assumed grid architecture can be viewed as a generalization of both paradigms, that is, if the number of columns in the grid is one, it reduces to the task parallelism paradigm, and if the number of rows in the grid is one, it reduces to the data parallelism paradigm. By the hybrid distribution, the communication overhead for moving the database is reduced, as the database partitions only need to be moved along the columns of the processor

grid instead of the whole grid. HD can also switch automatically to CD in later passes to further reduce communication overhead.

3.2.4 Discussion

Both data and task paradigms have advantages and disadvantages. They are appropriate for certain situations. The data parallelism paradigm has simpler communication and thus less communication overhead, it only needs to exchange the local counts of all candidates in each iteration. The basic count distribution algorithm CD can be further improved by using the hash techniques (PDM), candidate pruning techniques (DMA) and short-circuited counting (CCPD). However, the data parallelism paradigm requires that all the candidates fit into the main memory of each processor. If in some iteration there are too many candidates to fit into the main memory, all algorithms based on the data parallelism will not work (except SH) or their performance will degrade due to insufficient main memory to hold the candidates. SH tries to solve the insufficient main memory by spooling the candidates to disk (called a run in SH) when there is insufficient main memory. One possible problem with SH will be that there may be too many runs on disk, thus summing up the local counts in all runs will introduce a lot of disk I/O. Another problem associated with SH is the computation overhead to generate the candidates on the fly, as it needs to check whether all the *k-1*subsets of the *k*-itemsets in each transaction are large or not by looking up the bitmap of the *k-1* large itemsets, while Apriori only checks the itemsets in the join of two L_{k-1}.

The task parallelism paradigm was initially proposed to efficiently utilize the aggregate main memory of a parallel computer. It partitions and distributes the candidates among the processors in each iteration, so it utilizes the aggregate main memory of all processors and may not have the insufficient main memory problem with the number of processors increasing. Therefore, it can handle the mining problem with a very low minimum support. However, the task parallelism paradigm requires movement of the database partitions in addition to the large itemset exchange. Usually the database to be mined is very large, so the movement of the database will introduce tremendous communication overhead. Thus, it may be a problem when the database is very large. In the basic data distribution algorithm, as the database partition on each processor is broadcasted to all others, the total message for database movement is $O(p^2)$,

where p is the number of processors involved. IDD assumes a ring architecture and the communication is done simultaneously between the neighbors, so the total message is O(p). HPA uses a hash technique to direct the movement of the database partitions, that is, it only moves the transactions(precisely the subsets of transactions) to the appropriate destination processor which has the candidates. As the candidates are partitioned by a hash function, the subsets of the transactions are also stored by the same hash function. So the total message is reduced to O(p).

The performance studies in [Agrawal1996] [Park1995a] [Cheung1996] [Cheung1998], [Zaki1996] [Han1997] [Shintani1996] for both paradigms show that the data parallelism paradigm scales linearly with the database size and the number of processors. The task parallelism paradigm doesn't scale as well as the data parallelism paradigm but can efficiently handle the mining problem with lower minimum support, which can not be handled by the data parallelism paradigm or will be handled very insufficiently.

The performance studies in [Han1997] show that the hybrid distribution(HD) has speedup close to that of CD. This result is very encouraging, as it shows the potential to mine very large databases with a large number of processors.

3.2.5 Future of Parallel Algorithms

A promising approach is to combine the two paradigms. The hybrid distribution (HD) [Han1997] is more scalable than the task parallelism paradigm and has lessened the insufficient main memory problem. All parallel algorithms for mining association rules are based on the serial algorithm Apriori. As Apriori has been improved by many other algorithms, especially in reducing the number of database scans, parallelizing the improved algorithms is expected to deliver a better solution.

4 CLASSIFICATION AND COMPARISON OF ALGORITHMS

To differentiate the large number of algorithms, in the section we provide both a classification scheme and a qualitative comparison of the approaches. The classification scheme provides a framework which can be used to highlight the major differences among association rule algorithms (current and future). The qualitative comparison provides a high level performance analysis for the currently proposed algorithms.

4.1 Classification

In this subsection we identify the features which can be used to classify the algorithms. The approach we take is to categorize the algorithms based on several basic dimensions or features that we feel best differentiate the various algorithms. In our categorization we identify the basic ways in which the approaches differ. Our classification uses the following dimensions (summarized in Table 5):

1. **Target**: Basic association rule algorithms actually find all rules with the desired support and confidence thresholds. However, more efficient algorithms could be devised if only a subset of the algorithms were to be found. One approach which has been done to do this is to add constraints on the rules which have been generated. Algorithms can be classified as complete (All association rules which satisfy the support and confidence are found), constrained (Some subset of all the rules are found, based on a technique to limit them), and qualitative (A subset of the rules are generated based on additional measures, beyond support and confidence, which need to be satisfied).
2. **Type**: Here we indicate the type of association rules which are generated (for example regular (Boolean), spatial, temporal, generalized, qualitative, etc.)
3. **Data type**: Besides data stored in a database, association rules of a plain text might be very important information to find out. For example, "data", "mining" and "decision" may be highly dependent in an article of knowledge discovery.
4. **Data source**: Besides market basket data, association rules of data absent in the database might play significant role for decision purposes of a company.
5. **Technique**: All approaches to date are based on first finding the large itemsets. There could, of course, be other techniques which don't require that large itemsets first be found. Although to date we are not aware of any techniques which do not generate large itemsets, certainly this possibility does exist with the potential of improved performance. However, [Aggrawal1998c] proposed "strongly collective itemsets" to evaluate and find itemsets. The term "support" and "confidence" are quite difference from large itemset approach. An itemset I is said to be "strongly collective" at level K if the collective strength C(K) of I as well as any subset of I is at least K.
6. **Itemset Strategy**: Different algorithms look at the generation of items differently. This feature indicates how the algorithm looks at transactions as well as when the itemsets are

generated. One technique, Complete, could generate and count all potential itemsets. The most common approach is that introduced by (and thus called here) Apriori. With this strategy, a set of itemsets to count is generated prior to scanning the transactions. This set remains fixed during the process. A dynamic strategy generates the itemsets during the scanning of the database itself. A hybrid technique generates some itemsets prior to the database scan, but also adds new itemsets to this counting set during the scan.

7. **Transaction Strategy**: Different algorithms look at the set of transactions differently. This feature indicates how the algorithm scans the set of transaction. The complete strategy examines all transactions in the database. With the sample approach, some subset of the database (sample) is examined prior to processing the complete database. The partition techniques divide the database into partitions. The scanning of the database requires that the partitions be examining separately and in order.
8. **Itemset Data Structure**: As itemsets are generated, different data structures can be used to keep track of them. The most common approach seems to be a hash tree. Alternatively, a trie or lattice may be used. At least one technique proposes a virtual trie structure where only a portion of the complete trie is actually materialized.
9. **Transaction Data Structure**: Each algorithm assumes that the transactions are stored in some basic structure, usually a flat file or a TID list.
10. **Optimization:** Many recent algorithms have been proposed which improve on earlier algorithms by applying an optimization strategy. Various strategies have looked at optimization based on available main memory, whether or not the data is skewed, and pruning of the itemsets to be counted.
11. **Architecture**: As we have pointed out, some algorithms have been designed as sequential function in a centralized single processor architecture. Alternatively, algorithms have been designed to function in a parallel manner suitable for a multiprocessor or distributed architecture.
12. **Parallelism Strategy**: Parallel algorithms can be further described as task or data parallelism

Table 5 Classification

DIMENSION	VALUES
Target	Complete, Constrained, Qualitative
Type	Regular, Generalized, Quantitative, etc.
Data type	Database Data, Text
Data source	Market Basket, Beyond Basket
Technique	Large Itemset, Strongly Collective Itemset
Itemset Strategy	Complete, Apriori, Dynamic, Hybrid
Transaction Strategy	Complete, Sample, Partitioned
Itemset Data Structure	Hash Tree, Trie, Virtual Trie, Lattice
Transaction Data Structure	Flat File, TID
Optimization	Memory, Skewed, Pruning
Architecture	Sequential, Parallel
Parallel Strategy	None, Data, Task

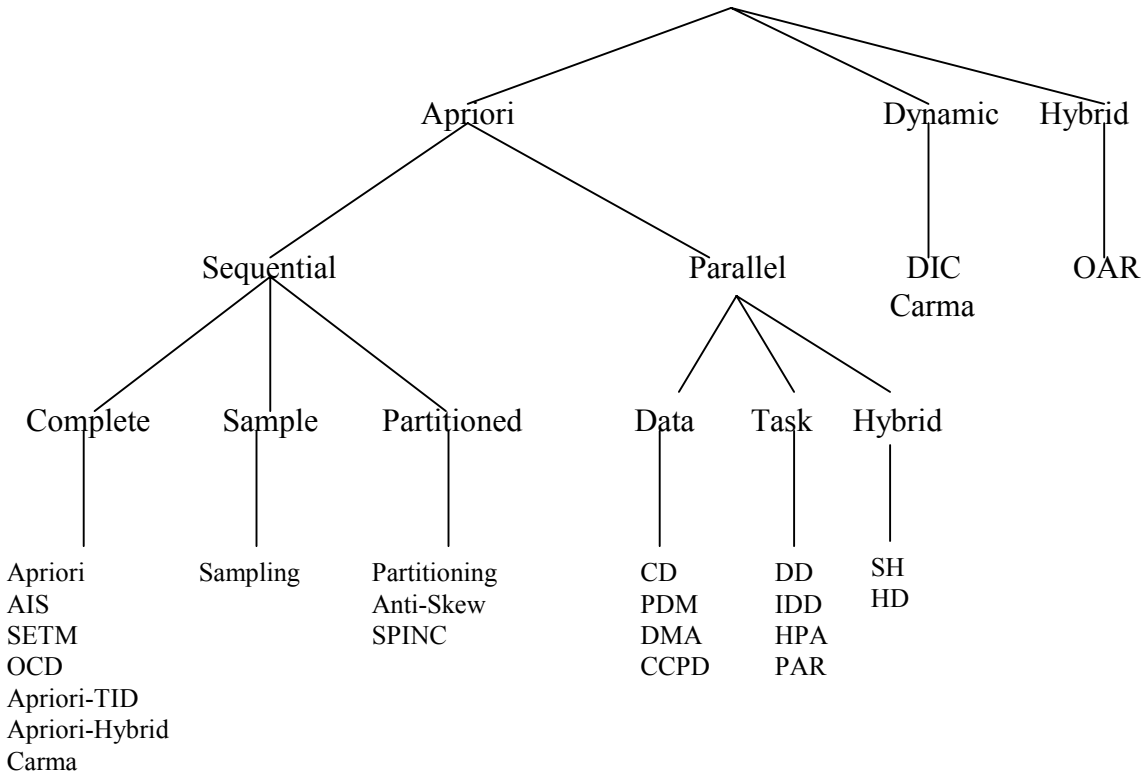

Figure 5 Classification of Complete, Regular, Itemset Algorithms

Figure 5 shows the classification of some algorithms we have seen so far. We only show those that generate large itemsets, and are complete and regular. Although all classification dimensions apply to all types of algorithms, to simplify the viewing, this figure only shows the major dimensions required to distinguish most algorithms. Algorithms are shown in the leaf nodes.

4.2 Comparing Algorithms

Here we compare the various algorithms based upon several metrics. Space requirements can be estimated by looking at the maximum number of candidates being counted during any scan of the database. We can estimate the time requirements by counting the maximum number of database scans needed (I/O estimate) and the maximum number of comparison operations (CPU estimate). Since most of the transaction databases are stored on secondary disks and I/O overhead is more important than CPU overhead, we focus on the number of scans in the entire database. Obviously, the worst case arises when each transaction in the database has all items. Let m be the number of items in each transaction, and L_k the large itemsets with k-items in a database D. Obviously, the number of large itemsets is 2^m. In level-wise techniques (e.g., AIS, SETM, Apriori), all large itemsets in L_1 are obtained during the first scan of the database. Similarly, all large itemsets in L_2 are obtained during the second scan, and so on. The only itemset in L_m is obtained during the m^{th} scan. All algorithms terminate when no additional entries in the large itemsets are generated, so an extra scan is needed. Therefore, the entire database will be scanned at most (m+1) times. Here it can be recalled that Apriori-TID scans the entire database in the first pass. Then it uses $\overline{C_k}$ rather than the entire database in the $(k+1)^{th}$ pass. However, that does not help at all in the worst case. The reason is that $\overline{C_k}$ will contain all of the transactions along with their items during the entire process. On the other hand, the OCD technique scans the entire database only once at the beginning of the algorithm to obtain large itemsets in L_1. Afterwards, OCD and Sampling use only a part of the entire database and the information obtained in the first pass to find the candidate itemsets of L_k where $1<k\leq m$. In the second scan they compute support of each candidate itemset. Therefore, there will be 2 scans in the worst case given enough main memory. The PARTITION technique also reduces the I/O overhead by reducing the number of database scans to 2. Similarly, CARMA needs at most 2 database scans.

The goodness of an algorithm depends on the accuracy of the number of "true" candidates it develops. As we have mentioned earlier, all algorithms use large itemsets of previous pass(es) to generate candidate sets. Large itemsets of previous itemsets are brought into the main memory to generate candidate itemsets. Again, candidate itemsets are needed to be in the main memory to obtain their support counts. Since enough memory may not be available, different algorithms propose different kinds of buffer management and storage structures. AIS proposed that L_{k-1} can be disk-resident if needed. SETM suggested that if $\overline{C_k}$ is too large to fit into main memory, write it to disk in FIFO manner. The Apriori family recommended to keep L_{k-1} on disk and bring into the main memory one block at a time to find C_k. However, $\overline{C_k}$ should be in the main memory to obtain support count in both Apriori-TID and Apriori-Hybrid. On the other hand, all other techniques assumed that there is enough memory to handle these problems. All other sequential techniques (PARTITION, Sampling, DIC and CARMA) consider a suitable part of the entire database which can fit in the main memory. The family of Apriori proposed suitable data structures (hash tree or array) for large itemsets as well as candidate sets which are presented in Table 6. However, neither AIS nor SETM proposed any storage structures.

Most commercially available implementations to generate association rules rely on the use of the Apriori technique.

Some algorithms are more suitable for use under specific conditions. AIS does not perform well when the number of items in the database is large. Therefore, AIS is more suitable in transaction databases with low cardinality. As we have mentioned earlier, Apriori needs less execution time than Apriori-TID in earlier passes. On the other hand, Apriori-TID outperforms Apriori in later passes. Therefore, Apriori-Hybrid shows excellent performance with proper switching. However, switching from Apriori to Apriori-TID is very crucial and expensive. Although OCD is an approximate technique, it is very much effective in finding frequent itemsets with lower threshold support. CARMA is an online user interactive feedback oriented technique which is best suited where transaction sequences are read from a network.

Table 6 summarizes and provides a means to briefly compare the various algorithms. We include in this table the maximum number of scans, data structures proposed, and specific comments.

Table 6 Comparison of Algorithms

Algorithm	Scan	Data structure	Comments
AIS	m+1	Not Specified	Suitable for low cardinality sparse transaction database; Single consequent
SETM	m+1	Not Specified	SQL compatible
Apriori	m+1	L_{k-1} : Hash table C_k: Hash tree	Transaction database with moderate cardinality; Outperforms both AIS and SETM; Base algorithm for parallel algorithms
Apriori-TID	m+1	L_{k-1} : Hash table C_k: array indexed by TID $\overline{C_k}$: Sequential structure ID: bitmap	Very slow with larger number of $\overline{C_k}$; Outperforms Apriori with smaller number of $\overline{C_k}$;
Apriori-Hybrid	m+1	L_{k-1} : Hash table 1st Phase: C_k: Hash tree 2nd phase: C_k: array indexed by IDs $\overline{C_k}$: Sequential structure ID: bitmap	Better than Apriori. However, switching from Apriori to Apriori-TID is expensive; Very crucial to figure out the transition point.
OCD	2	Not specified	Applicable in large DB with lower support threshold.
Partition	2	Hash Table	Suitable for large DB with high cardinality of data; Favors homogenous data distribution
Sampling	2	Not Specified	Applicable in very large DB with lower support.
DIC	Depends on interval size	Trie	Database viewed as intervals of transactions; Candidates of increased size are generated at the end of an interval
CARMA	2	Hash Table	Applicable where transaction sequences are read from a Network; Online, users get continuous feedback and change support and/or confidence any time during process.
CD	m+1	Hash table and tree	Data Parallelism.
PDM	m+1	Hash table and tree	Data Parallelism; with early candidate pruning
DMA	m+1	Hash table and tree	Data Parallelism; with candidate pruning

Table 6 (cont'd) Comparison of Algorithms

Algorithm	Scan	Data structure	Comments
CCPD	m+1	Hash table and tree	Data Parallelism; on shared-memory machine
DD	m+1	Hash table and tree	Task Parallelism; round- robin partition
IDD	m+1	Hash table and tree	Task Parallelism; partition by the first items
HPA	m+1	Hash table and tree	Task Parallelism; partition by hash function
SH	m+1	Hash table and tree	Data Parallelism; candidates generated independently by each processor.
HD	m+1	Hash table and tree	Hybrid data and task parallelism; grid parallel architecture

5 EXTENDED ASSOCIATION RULES

Association rule algorithms presented in previous sections generate all association rules satisfying given confidence and support requirements. There have been algorithms which either generate rules under other requirements or extend the basic definition of what an association rule is. We examine the body of work to extend the basic algorithms in this section.

5.1 Generalized Association Rules

Generalized association rules use the existence of a *hierarchical taxonomy (concept hierarchy)* of the data to generate different association rules at different levels in the taxonomy [Srikant1995]. Figure 6 shows an example of a taxonomy on market basket data. Here beverage is further divided into coffee, tea, soft drinks, and juice. Juice is divided into orange, apple, cranberry, and grape. When association rules are generated, we could generate them at any of the hierarchical levels present. As would be expected, when rules are generated for items at a higher level in the taxonomy, both the support and confidence increase. In a given transaction database, there may be multiple taxonomies for different items and even multiple taxonomies for the same item. A *generalized association rule*, $X \Rightarrow Y$, is defined identically to that of regular association rule, except that no item in Y can be an ancestor of any in X [Srikant1995]. An *ancestor* of an item is one which is above it in some taxonomy. A supermarket may want to find associations relating to soft drinks in general or may want to identify those for a specific brand or type of soft

drink (such as a cola). The generalized association rules allow this to be accomplished and also ensure that all association rules (even those across levels in different taxonomies are found.

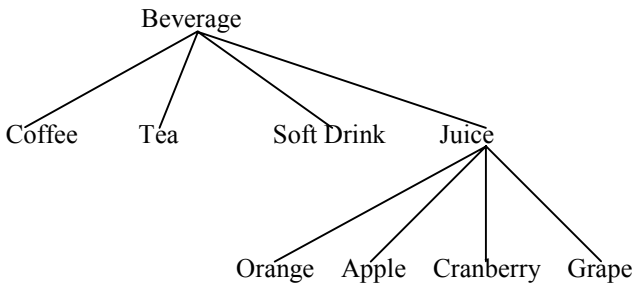

Figure 6 Market Basket Taxonomy

The generalized association rule problem is to generate association rules for all levels of all taxonomies. One approach to do this would be to take each transaction and expand each item to include all items above it in any hierarchy [Srikant1995]. This naïve approach is quite expensive and other more efficient algorithms have been proposed. One algorithm, Cumulate, uses several optimization strategies to reduce the number of ancestors which need to be added to each transaction [Srikant1995]. Another approach, Stratification, counts itemsets by their levels in the taxonomy and uses relationships about items in a taxonomy to reduce the number of items to be counted [Srikant1995]. Several parallel *algorithms* to generate generalized association rules have also been proposed [Shintani1998]

When association rules are generated from across different levels in the concept hierarchy, they are called *multiple-level association rules* in [Han1995]. The approach here is to generate large itemsets in a top-down fashion on the concept hierarchy using an Apriori type algorithm. The notation L[i,j] indicates the level-i (in the concept hierarchy) large-j itemsets. Only the children of large j-itemsets at level i are considered to be candidates for large j-itemsets at level i+1. To aid in this process L[i,j] is used to remove small items and transactions with only small items.

5.2 Temporal and Spatial Association Rules

Spatial databases contain location information concerning the data being stored. This may be in the form of latitude-longitude pairs, street addresses, zip codes, or other geographic data. While spatial data mining examines the same types of problems as traditional data mining, problem statements and potential solutions may be tailored to the fact that spatial data is involved. For example, spatial operations (within, near, next to, etc.) can be used to describe relationships among tuples in the database. A *spatial association rule*, $X \Rightarrow Y$, is an association rule where both X and Y are sets of predicates, some of which are spatial [Koperski1995]. A spatial association rule holds for a tuple, T, in a database if both predicates, X and Y, are true for T. Definitions for confidence and support are identical to those for regular association rules. Suppose that a database contains information about public schools in a particular county. This database contains data about public facilities (parks, schools, municipal buildings, etc.), geographic features (rivers, lakes, etc.), private buildings, and public infrastructure (roads, bridges, etc). The following is a spatial association rule:

$$\text{Elementary}(T) \wedge \text{Near}(T,\text{housing development}) \Rightarrow \text{Adjacent to } (T,\text{park})$$

This rule indicates that an elementary school which is near a housing development is also adjacent to a park. Unlike market basket data, we may need to look at other tuples outside the one being examined, to determine the validity of a spatial association rule. We only need to look at a tuple T to see if it is an elementary school as this will be shown in the value for some attributes. However interpreting the truth of the spatial predicates (near and adjacent to in this case) may require looking at other tuples in the database (or other databases). Thus determining the truth of a spatial predicate may be quite difficult and expensive. One approach to improve the efficiency of mining spatial association rule is a two step technique where the first step examines approximate satisfaction of spatial predicates by using a coarse interpretation of the spatial relationships [Koperski1995]. This step serves as a filtering process which can drastically improve the second step which examines an exact matching of the predicate. The use of dedicated spatial data structures including R-trees and MBR representations of the spatial features also improves performance.

Temporal association rules are similar to spatial except that the predicates involve time. Similarly, *spatial-temporal association rules* involve both time and space predicates.

5.3 Quantitative Association Rules

Most initial research into association rules has assumed that the data is categorical. The *quantitative association rule* problem assumes that data may be both categorical and quantitative [Srikant1996b]. By dividing quantities into sets of intervals, rules can be derived based on these. The following is an example of a quantitative association rule:

Customer pays between \$3 and \$5 for bread ⇒ Pays between \$10 and \$20 for wine.

For qualitative attributes, the values of the attributes are mapped to a set of consecutive integers. Quantitative attributes can be partitioned into intervals as well as non-partitioned. In case of non-partitioned quantitative attributes, the values are mapped to consecutive integers such that the order of the values is preserved. On the other hand, intervals of partitioned quantitative attributes are mapped to consecutive integers such that the order of the intervals is preserved.

Table Name: Person

ID	Age	Married	NumCar
1000	21	No	0
2000	23	Yes	1
3000	24	No	1
4000	26	Yes	2
5000	29	Yes	2

Figure 7 Example Table "Person" for Quantitative Attribute [source: Srikant1996a]

Figure 7 shows an example table "Person" with three attributes. Age and NumCar are quantitative attributes, where Married is a qualitative attribute. It is mentioned in [[Cengiz1997], Srikant1996a]] that if the quantitative rules problem can be mapped to the Boolean rules problem, any algorithm for finding *Boolean association rules* (or regular) can be used to find quantitative association rules.

ID	Age: 20..24	Age: 25..29	Married: Yes	Married: No	NumCar : 0	NumCar: 1	NumCar: 2
1000	1	0	0	1	1	0	0
2000	1	0	1	0	0	1	0
3000	1	0	0	1	0	1	0
4000	0	1	1	0	0	0	1
5000	0	1	1	0	0	0	1

Figure 8 Mapping to Boolean Association Rules Problem [source: Srikant1996a]

Figure 8 shows the mapping for the example given in Figure 7. This simple mapping approach leads to two problems. At first, if the number of values/intervals for an attribute is large, the support for any particular values/intervals can be low. This is called the "Minsup" problem. Secondly, if the number of values/intervals for an attribute is small, there is a possibility of losing information. That means some rules may not have threshold confidence. This problem is referred to as the "Minconf" problem. To overcome aforementioned problems, all possible ranges over values/intervals may be combined when processing each particular value/interval. That means, combining adjacent values/intervals to avoid the threshold support problem, and increasing the number of intervals to avoid the threshold confidence problem. However, this approach leads to two new problems (Higher Execution Time and Many Rules). High Execution time arises when the number of intervals for an attribute is increased. On the other hand, we will obtain increased number of rules (we might not be interested in some of them) if we consider any range that contains the interval satisfying the threshold support. To avoid the "MinSup" problem, [Srikant1996a] considered ranges over adjacent values/intervals of quantitative attributes. By introducing a user specified "maximum support" parameter, the extension of adjacent values/intervals is restricted. The adjacent values/intervals are combined until the combined support is less than the maximum support. However, any single interval/value whose support exceeds maximum support is still considered. As a result of this, the "Higher Execution Time" problem is reduced to a certain extent. In this technique, a database record is treated as a set of <attribute, integer value> pairs without loss of generality.

The problem of finding frequent itemsets from the database with quantitative attributes is solved in three steps. At first, decide whether each attribute is to be partitioned or not. If an attribute is to be partitioned, determine the number of partitions. Then, map the values of the attribute to a set of consecutive integers. Afterwards, find support of each value of all attributes. As mentioned earlier, to avoid "Minsup" problem, adjacent values are combined as long as their support is less than user-specified maximum support. All ranges and values with minimum support form the set of frequent itemsets. Assuming minimum support of 40%, some large itemsets with corresponding supports are given in Figure 9.

Itemset	Support
{<Age: 20…24>}	60%
{<Age: 25…29>}	40%
{<Married: Yes>}	60%
{<Married: No>}	40%
{<Numcar: 1>}	40%
{<Numcar: 2>}	40%
{<Age: 20…24> and <Married: No>}	40%
{<Age: 25…29> and <Married: Yes>}	40%

Figure 9 Some Large Itemsets of Items Given in Figure 7

[Srikant1996a] introduced the partial-completeness measure to decide whether an attribute is to be partitioned or not. Partial-completeness measure also assists in determining the number of partitions. The term "partial-completeness" is explained as follows. Let R be the set of rules obtained by considering all ranges over the raw values of quantitative attributes. Let R_1 be the set of rules obtained by considering all ranges over the partition of quantitative attributes. The information loss is measured by looking at how far apart the "closest" rules are in R_1 when we go from R to R_1. A close rule will be found if the minimum confidence level for R_1 is less than that for R by a certain amount. Mathematically, partial-completeness can be defined as follows: Let C denote the set of all large itemsets in database D. P is called K-complete with respect to C, if for every $X \in P$ there exists a generalization of X, called X′, in P such that:

$$\text{attributes}(X) \subseteq \textit{attributes}(X') \text{ and } \text{Support}(X') \leq K * \text{support}(X), \text{ where } K \geq 1$$

It is mentioned in [Srikant1996a] that for any rule $X \Rightarrow Y$, there is a rule $X' \Rightarrow Y'$ where, X' and Y' are generalization of X and Y, respectively, and the support of $X' \Rightarrow Y'$ is almost K times the support $X \Rightarrow Y$. It is also found that equi-depth partitioning minimizes the required number of partitions. The number of intervals is computed as follows:

$$\text{No of Intervals} = \frac{2n}{m \times (K-1)}$$

Here, n= Number of quantitative attributes, m= Minimum support, and K= Partial Completeness level = $1+\frac{2n(\text{maxsup})}{\text{minsup}}$.

A special type of quantitative association rules is called *profile association rules* [Aggarwal1998a, Aggarwal1998b]. Here customer profile data is present on the left hand side of the association rule statement, while the right hand side contains information about the customer behavior. Below is an example of a profile association rule

$$\text{Income} > \$200{,}000 \Rightarrow \text{Home purchase} > \$400{,}000$$

Here we indicate that if an individual has an income over $200,000, then he will purchase a home worth over $400,000. By linking customer profiles with buying information, a company can target marketing campaigns based on the demographics stated by the profile (left hand side of the association rule). With profile rules, the hierarchical nature of profiles (for example income between $200,000 and $300,000 is a further refinement of the above profile) can be used to reduce the overhead of generating itemsets. This is similar to the use of the concept hierarchy in generalized rules. This partitioning of the data may be stored in a special index called an S-Tree which is similar to and R-Tree. An interesting rule tree is used to store the rules based on the hierarchical nature of the profiles.

Another type of quantitative rule is called a *ratio rule* [Korn1998]. These rules identify relationships between attributes which satisfy a ratio. For example:

$$\text{Income: Home purchase is 1:2}$$

Notice that the profile association rule above satisfies this ratio restriction. Ratio rules may be across multiple attributes (not just two).

The standard quantitative association rules assume that the range of data is partitioned into precise discrete regions. However, the partitioning into more fuzzy regions could yield other interesting associations. When this is done, the rules are called *fuzzy association rules*

[Kuok1998]. Suppose that home prices were divided into three discrete regions [0-$99,999], [$100,000, $299,999], [$300,000,∞). With the normal approach a house would only appear in one of these three regions. Even though a house might cost $299,999 a transaction with this value in it would not be considered in rules for house in the third region. A fuzzy association rule is of the following form.

$$X \text{ is } A \Rightarrow Y \text{ is } B$$

Here X and Y are itemsets while A and B are fuzzy set membership functions for the corresponding attributes in X and Y. To say that X is A is satisfied means that the sum of the fuzzy membership votes is above a certain threshold [Kuok1998]. The sum of the membership votes for A divided by the total number of records is called the *significance.*

5.4 Interval Data Association Rules

In [Srikant1996a], it is observed that **t**he complexity of the search in the formulation of the association rule problem does not only depend on the number of attributes but also on the number of values of an attribute. The complexity in mining association rules in relational tables with large domains is reduced by grouping data together and considering collectively. If an attribute is linearly ordered then values may be grouped into ranges [Srikant1996a]. For example, age can be partitioned into ranges (e.g. ranges of 5 years increments) instead of considering all values of an age attribute. Solutions presented in [Srikant1996a] do not work well when applied to interval data where separation between data values has meaning [Miller1997]. Consider the example given in Figure 10 and an interval of 20K. We see that there are some unnecessary intervals which do not contain any values. This problem can be overcome by using Equi-depth Interval mentioned in [Srikant1996a]. In this method the depth (support) of each partition is determined by the partial completeness level. Therefore, the intervals are determined by their relative ordering. For a depth d, the first d values (in order) are placed in one interval, the next d in the second interval, etc. The density of an interval or the distance between intervals is not considered. Distance-based interval is introduced in [Miller1997]. This technique is based on the idea that intervals that include close data values (e.g. [81K, 81K]) are more meaningful than intervals involving distant values (e.g. [31K, 80K]).

Salary	Interval 18K..38K	Interval 38K..58K	Interval 58K..78K	Interval 78K..98K
18K	1	0	0	0
30K	1	0	0	0
31K	1	0	0	0
80K	0	0	0	1
81K	0	0	0	1
82K	0	0	0	1

Figure 10 Partitioning Salary into Several Intervals [source: Miller1997]

5.5 Multiple Min-supports Association Rules

Previous work surveyed has focused on mining association rules in large databases with single support. Since a single threshold support is used for the whole database, it assumes that all items in the data are of the same nature and/or have similar frequencies. In reality, some items may be very frequent while others may rarely appear. However, the latter may be more informative and more interesting than the earlier. For example, besides finding a rule bread $\Rightarrow$ cheese with a support of 8%, it might be more informative to show that wheatBread $\Rightarrow$ swissCheese with a support of 3%. Another simple example could be some items in a super market which are sold less frequently but more profitable, food processor and cooking pan [Liu1999]. Therefore, it might be very interesting to discover a useful rule foodProcessor $\Rightarrow$ cookingPan with a support of 2%.

If the threshold support is set too high, rules involving rare items will not be found. To obtain rules involving both frequent and rare items, the threshold support has to be set very low. Unfortunately, this may cause combinatorial explosion, producing too many rules, because those frequent items will be associated with another in all possible ways and many of them are meaningless. This dilemma is called the "rare item problem" [Liu1999]. To overcome this problem, one of the following strategies may be followed [Han1995] [Liu1999]: (a) split the data into a few blocks according to the supports of the items and then discover association rules in each block with a different threshold support, (b) group a number of related rare items together into an abstract item so that this abstract item is more frequent. Then apply the algorithm of finding association rules in numerical interval data.

It is evident that both approaches are ad hoc and approximate. Rules associated with items across different blocks are difficult to find using the first approach. The second approach cannot discover rules involving individual rare items and the more frequent items. Therefore, a single threshold support for the entire database is inadequate to discover important association rules because it cannot capture the inherent natures and/or frequency differences in the database. [Liu1999] extended the existing association rule model to allow the user to specify multiple threshold supports. The extended new algorithm is named as MISapriori. In this method, the threshold support is expressed in terms of *minimum item supports (MIS)* of the items that appear in the rule. The main feature of this technique is that the user can specify a different threshold item support for each item. Therefore, this technique can discover rare item rules without causing frequent items to generate too many unnecessary rules.

Similar to conventional algorithms, the MISapriori generates all large itemsets by making multiple passes over the data. In the first pass, it counts the supports of individual items and determines whether they are large. In each subsequent pass, it uses large itemsets of the previous pass to generate candidate itemsets. Computing the actual supports of these candidate sets, the MISaprioi determines which of the candidate sets are actually large at the end of the pass. However, the generation of large itemsets in the second pass differs from other algorithms. A key operation in the MISapriori is the sorting of the items I in ascending order of their MIS values. This ordering is used in the subsequent operation of the algorithm.

The extended model was tested and evaluated by using synthetic data as well as real-life data sets. In the experimental study of this algorithm with synthetic data, three very low LS values, 0.1%, 0.2%, and 0.3% were used. It has been reported that the number of large itemsets is significantly reduced by MISapriori method when α is not too large. The number of large itemsets found by this approach is close to single minsup method when α becomes larger. This is because when α becomes larger more and more items' MIS values reach LS. It has also been argued that the execution time reduces significantly.

5.6 Multimedia Association Rules

Although multimedia databases have become one of the most promising research areas in the database community, discovering association rules in multimedia databases has not received

much attention. Many relational and object oriented databases have been using multimedia objects more frequently than the previous years [Zaiane1998]. These multimedia objects include photo, video, audio, etc. Tremendous use of the global internet has increased the demand of multimedia objects. It is a necessity to extract these data and find associations among them. An example of association rule can be:

Image related to ocean $\wedge$ size is big => Color is blue

More research is needed in this area.

5.7 Maximal Association Rules

Maximal association rules allow a stronger statement of association between sets of attributes than is facilitated with the use of regular association rules. For example, a rule of the form Butter $\Rightarrow$ Newspaper indicates that when Butter appears in a transaction so does Newspaper with some confidence and support. This relationship would be true in a transaction consisting of t={Newspaper, Butter, Eggs} as well as in a transaction containing only u={Butter, Newspaper}. Given two itemsets X and L where X$\subseteq$L, X:L is said to be *maximal* with respect to a tuple t if t$\cap$L=X. An association rule X $\Rightarrow$ Y is a *maximal association rule* if X and Y are maximal sets [Feldman1997b]. Suppose that there are two categories of items sold at a store: food and miscellaneous items. Butter and Eggs are categorized as food, while Newspaper is in the miscellaneous category. Here Butter and Newspaper are both maximal in u, but only Newspaper is maximal in t. Thus Butter $\Rightarrow$ Newspaper holds in u but not t. Notice that this support would be 100% if it were a regular association rule (since it holds for both transactions), but only 50% as a maximal rule.

5.8 Constraints on Rules

Many algorithms have been proposed to reduce the total number of itemsets generated based on constraints on the resulting rules. Certainly the support and confidence values put constraints on the generated rules. Proposed constraints include the following:

a) Recently rules have been defined to be important is they are *interesting.* Interesting rules are those which have a greater than expected support and confidence when compared to

what they would be if attributes occurred randomly [Aggarwal1998a] [Tsur1998] [Srikant1996c].

b) Another measure of goodness has been defined based on a root-mean-square *guessing error* [Korn1998]. The guessing error for a specific transaction and attribute within the transaction is calculated as the difference between the actual value and an estimate of that value. The overall guessing error is then the root-mean-square of all the guessing errors.

c) The chi-squared *correlation* test has been proposed to measure association rules [Brin1997b]. This measure is applicable to generalized association rules involving a lattice of subsets.

5.9 SQL Extensions

There have been several proposed SQL extensions to facilitate the generation of association rules. One approach assumes that temporary tables are created [Houtsma1995] [Houtsma1996]. A new MINE RULE operator is proposed in [Meo1996]. In use, this operator precedes an SQL SELECT statement. The SELECT statement is terminated with an EXTRACTING RULES clause which includes the SUPPORT and CONFIDENCE values requested.

6 MAINTENANCE OF DISCOVERED ASSOCIATION RULES

Most association rule algorithms assume a static database. With these approaches the algorithm must be performed completely against each new database state to be able to generate the new set of association rules. In large databases or volatile databases, this may not be acceptable. There have been many proposals to facilitate the maintenance of association rules. These approaches are often referred to as *incremental updating* strategies when only additions to the transaction database are considered.

The first incremental updating strategy was call *Fast Update (FUP)* [Cheung1996a]. The problem with incremental updating is to find the large itemsets for a database D$\cup$*db* where both D and *db* are sets of transactions and the set of large itemsets, L, for D is already known. FUP is based on the Apriori algorithm. For each iteration, only *db* is scanned using the known set of large itemsets of size k, L_k, from D as the candidates. This is used to remove the candidates

which are no longer large in the larger database, D∪*db*. Simultaneously a set of new candidates is determined. Three variations of this algorithm have subsequently been proposed which create less candidates, *FUP** and FUP_2, and are applicable for multi-level association rules, *MLUp* [Cheung1996b] [Cheung1997].

Another approach to maintaining association rules is based on the idea of sampling [Lee1997a]. The algorithm proposed in this paper, *Difference Estimations for Large Itemsets (DELI),* uses sampling to estimate the upper bound on the difference between the old and new sets of association rules. Small changes to the association rule set are ignored. Performance studies showed the effectiveness of the DELI approach in saving resources.

A third approach determines the large itemsets of the incremental database and only scans the original database if the negative border of the large itemsets expands from that of the original database [Thomas1997]. In this situation only one scan over the original database is then required to find all large itemsets.

7 SUMMARY

Mining Association Rules is one of the most used functions in data mining. Association rules are of interest to both database researchers and data mining users. We have provided a survey of previous research in the area as well as provided a brief classification strategy and comparison of approaches.

8 BIBLIOGRAPHY

[Aggarwal1998a] Charu C. Aggarwal, Zheng Sun, and Philip S. Yu, Online Algorithms for Finding Profile Association Rules, *Proceedings of the ACM CIKM Conference,* 1998, pp 86-95.

[Aggarwal1998b] C. C. Aggarwal, J. L. Wolf, P. S. Yu, and M. Epelman, Online Generation of Profile Association Rules, *Proceedings of the International conference on Knowledge Discovery and Data Mining,* August 1998.

[Aggrawal1998c] Charu C. Aggarwal, and Philip S. Yu, A New Framework for Itemset Generation, *Principles of Database Systems (PODS)* 1998, Seattle, WA.

[Agrawal1993] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami, Mining Association Rules Between Sets of Items in Large Databases, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pp. 207-216, Washington, D.C., May 1993.

[Agrawal1993a] Rakesh Agrawal, Tomasz Imielinski and Arun N. Swami", Data Mining: A Performance perspective, *IEEE Transactions on Knowledge and Data Engineering,* Vol. 5, No. 6, December 1993, pp. 914-925.

[Agrawal1994] Rakesh Agrawal and Ramakrishnan Srikant, Fast Algorithms for Mining Association Rules in Large Databases, *Proceedings of the Twentieth International Conference on Very Large Databases*, pp. 487-499, Santiago, Chile, 1994.

Rakesh Agrawal and Ramakrishnan Srikant, Mining Sequential Patterns, *Proceedings of the 11th IEEE International Conference on Data Engineering*, Taipei, Taiwan, March 1995, IEEE Computer Society Press.

Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, Hannu Toivonen, and A. Inkeri Verkamo, Fast Discovery of Association Rules, In Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pp. 307-328, Menlo Park, CA, 1996. AAAI Press.

[Agrawal1996] Rakesh Agrawal and John C. Shafer, Parallel Mining of Association Rules, *IEEE Transactions on Knowledge and Data Engineering,* Vol. 8, No. 6, pp. 962-969, December 1996.

Roberto J. Bayardo Jr., Rakesh Agrawal, Dimitris Gunopulos, Constarint-Based Rule Mining in Large, Dense Databases, *Proceedings of the 15th International Conference on Data Engineering*, 23-26 March 1999, Sydney, Australia, pp.188-197

[Brin1997a] Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman, and Shalom Tsur, Dynamic Itemset Counting and Implication Rules for Market Basket Data, *Proceedings of the ACM SIGMOD Conference*, pp. 255-264, 1997.

[Brin1997b] Sergey Brin, Rajeev Motwani, and Craig Silverstein, Beyond Market Baskets: Generalizing Association Rules to Correlations, *Proceedings of the ACM SIGMOD Conference*, pp. 265-276, 1997.

[Cengiz1997] Ilker Cengiz, Mining Association Rules, Bilkent University, Department of Computer Engineering and Information Sciences, Ankara, Turkey, 1997, URL: *http://www.cs.bilkent.edu.tr/~icegiz/datamone/mining.html*.

[Chat1997] Jaturon Chattratichat, John Darlington, Moustafa Ghanem, and et. al, Large Scale Data Mining: Challenges and Responses, *Proceedings of the 3th International Conference on Knowledge Discovery and Data Mining*, pp. 143-146, August 1997.

[Chen1996] Ming-Syan Chen, Jiawei Han and Philip S. Yu, Data Mining: An Overview from a Database Perspective, *IEEE Transactions on Knowledge and Data Engineering,* Vol. 8, No. 6, pp. 866-883, 1996.

[Cheung1996] David Wai-Lok Cheung, Jiawei Han, Vincent Ng, Ada Wai-Chee Fu, and Yongjian Fu, A Fast Distributed Algorithm for Mining Association Rules, *Proceedings of PDIS*, 1996.

[Cheung1996a] D. W. Cheung, J. Han, V. T. Ng, and C. Y. Wong, Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique*, Proceedings of the 12th IEEE International Conference on Data Engineering,* pp. 106-114, February 1996.

[Cheung1996b] David W. Cheung, Vincent T. Ng, and Benjamin W. Tam, Maintenance of Discovered Knowledge: A Case in Multi-level Association Rules, *Proceedings of the Second International KDD Conference, 1996, pp307-310.*

[Cheung1996c] David Wai-Lok Cheung, Vincent T. Ng, Ada Wai-Chee Fu, and Yongjian Fu, Efficient Mining of Association Rules in Distributed Databases, *IEEE Transactions on Knowledge and Data Engineering,* Vol. 8, No. 6, pp. 911-922, December 1996.

[Cheung1997] David Wai-Lok Cheung, Sau Dan Lee and Benjamin C. M. Kao, A General Incremental Technique for Maintaining Discovered Association Rules, pp. 185-194, *Proceedings of the DASFAA*, 1997.

[Cheung1998] David W. Cheung and Yongqiao Xiao, Effect of Data Skewness in Parallel Mining of Association Rules, *Proceedings of the 2nd Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 48-60, Melbourne, Australia, April 1998.

David W. Cheung, Kan Hu and Shaowei Xia, Asynchronous Parallel Algorithm for Mining Association Rules on a Shared Memory Multi-processors, *Proceedings of the tenth annual ACM symposium on Parallel algorithms and architectures*, June 28 - July 2, 1998, Puerto Vallarta, Mexico, 279-288

David W. Cheung, Jiawei Han, Vincent T. Ng and C. Y. Wong, Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique, *Proceedings of the Twelfth International Conference on Data Engineering*, February 26 - March 1, 1996, New Orleans, Louisiana, pp. 106-114.

C. Faloutsos, F. Korn, A. Labrinidis, Y. Kotidis. A. Kaplunovich and D. Perkovi'c, Quantifiable Data Mining Using Principle Component Analysis, *Technical Research Report, TR 97-25*, ISR, National Science Foundation.

[Fayyad1996] Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth, From Data Mining to knowledge Discovery: An Overview, *Advances in Knowledge Discovery and Data Mining,* Edited by Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padraic Smyth, and Ramasamy Uthurusamy, AAAI Press, 1996, pp 1-34.

R. Feldman, Y. Aumann, A. Amir, and H. Mannila, Efficient Algorithms for Discovering Frequent Sets in Incremental Databases, *Proceedings of the 1997 SIGMOD Workshop on DMKD,* May 1997. Tucson, Arizona.

[Feldman1997b] Ronen Feldman, Yonatan Aumann, Amihood Amir, Amir Zilberstein, and Willi Kloesgen, Maximal Association rules: a New Tool for Mining for Keyword Co-occurrences in Document Collections, *Poster Papers, Knowledge Discovery in Database (KDD),* 1997, Newport Beach, California, USA, pp.167-170.

[Fukuda1996a] Takeshi Fukuda, Yasuhiko Morimoto, Shinichi Morishita and Takeshi Tokuyama, Mining Optimized Association Rules for Numeric Attributes, *PODS '96*, pp. 182-191, Montreal, Quebec, Canada.

[Fukuda1996b] Takeshi Fukuda, Yasuhiko Morimoto, Shinichi Morishita and Takeshi Tokuyama, Data Mining Using Two-Dimensional Optimized Association Rules: Scheme, Algorithms, and Visualization, *ACM SIGMOD, June 1996,* Montreal, Canada. pp. 13-23.

[Fukuda1996c] Takeshi Fukuda, Yasuhiko Morimoto, Shinichi Morishita and Takeshi Tokuyama, (SONAR): System for Optimized Numeric Association Rules, *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, 4-6 June, 1996, pp. 553.

[Fukuda1996d] T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama, Mining Optimized Association Rules for Numeric Attributes, *Proceedings of the Fifteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp. 182-191, Montreal, Quebec, Canada, 3-5 June 1996.

[Fukuda1996e] Takeshi Fukuda, Yasuhiko Morimoto, Shinichi Morishita and Takeshi Tokuyama, Constructing Efficient Decision Trees by Using Optimized Numeric Association Rules, *Proceedings of the 22nd International Conference on Very Large Databases*, 1996, pp. 146-155.

[Goebel1999] Michael Goebel, and Le Gruenwald, A Survey of Data Mining and Knowledge Discovery Software Tools, *SIGKDD Explorations, ACM SIGKDD*, June 1999, Volume 1, Issue 1, pp. 20-33.

[Han1995] Jiawei Han and Yongjian Fu, Discovery of Multiple-Level Association Rules from Large Databases, *Proceedings of the 21nd International Conference on Very Large Databases*, pp. 420-431, Zurich, Swizerland, 1995.

[Han1997] Eui-Hong Han, George Karypis, and Vipin Kumar, Scalable Parallel Data Mining For Association Rules, *Proceedings of the ACM SIGMOD Conference*, pp. 277-288, 1997.

[Harada1998] Lilian Harada, Naoki Akaboshi, Kazutaka Ogihara, and Riichiro Take, Dynamic Skew Handling in Parallel Mining of Association Rules, *Proceedings of the 7th International Conference on Information and Knowledge Management*, pp.76-85, Bethesda, Maryland, USA, 1998.

[Hidb1999] Christian Hidber, Online Association Rule Mining, SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadephia, Pennsylvania, pp.145-156.

[Houtsma1995] M. Houtsma and A. Swami, Set-Oriented Mining for Association Rules in Relational Databases, *Proceedings of the 11th IEEE International Conference on Data Engineering*, pp. 25-34, Taipei, Taiwan, March 1995.

[Houtsma1996] M. A. W. Houtsma and A. Swami, Set-Oriented Mining in Relational Databases, *Data and Knowledge Engineering, 1996.*

[Imasaki] Kenji Imasaki, Implementation of Count Distribution and Eclat Algorithm a Network of Workstations: Parallel Algorithms and their Implementation (Project 2). *School of Computer Science, Carleton University, Canada*.

[Kamber1997] Micheline Kamber, Jiawei Han, and Jenny Y. Chiang, Metarul-Guided Mining of Multi-Dimensional Association Rules Using Data Cubes, *Poster Papers, KDD 1997*, Newport Beach, California, USA , pp. 207-210.

[Klemettinen1994] Mika Klemettinen, Heikki Mannila, Pirjo Ronkainen, Hannu Toivonen and A. Inkeri Verkamo, Finding Interesting Rules From Large Sets of Discovered Association Rules", *Proceedings of the Third International Conference on Information and Knowledge Management (CIKM'94)*, November 1994, pp. 401-407.

[Koperski1995] K. Koperski and J. Han, Discovery of Spatial Association Rules in Geographic Information Databases, *Lecture Notes in Computer Science*, Vol. 951, pp. 47-66, 1995.

[Korn1998] Flip Korn, Alexandros Labrinidis, Yannis Kotidis, and Christos Faloutsos, Ratio Rules: A New Paradigm for Fast, Quantifiable Data Mining, *Proceedings of the 24th VLDB Conference,* 1998.

[Kuok1998] Chan Man Kuok, Ada Fu, Man Hon Wong, Mining Fuzzy Association Rules in Databases, *ACM SIGMOD RECORD,* Vol 27, No 1, March 1998.

[Lee1997a] S.D. Lee and David W. Cheung, Maintenance of Discovered Association Rules: When to Update? *Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD)*, May 11, 1997, Tucson, Arizona

[Lee1998] S.D. Lee, David W. Cheung, and Ben Kao, Is Sampling Useful in Data Mining? A Case in the Maintenance of Discovered Association Rules, *Department of Computer Science*, The university of Hong Kong, Hong Kong.

[Lee1998a] S.D. Lee, David W. Cheung, and Ben Kao, Is Sampling Useful in Data Mining? A Case in the Maintenance of Discovered Association Rules, *International Journal on Data Mining and Knowledge Discovery*, Volume 2, Number 3, September 1998, pp. 233-262

[Lin1998] Jun-Lin Lin and M. H. Dunham, Mining Association Rules: Anti-skew Algorithms, *Proceedings of the 14th IEEE International Conference on Data Engineering*, Olando, Florida, February 1998.

[Liu1998] Bing Liu, Wynne Hsu and Yiming Ma, Integrating Classification and Association Rule Mining, *American Association of Artificail Intelligence*, *KDD*1998, pp. 80-86.

[Liu1999] Bing Liu, Wynne Hsu and Yiming Ma, Mining Association Rules with Multiple Supports, *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-99)*, August 15-18, SanDiego, CA, USA.

[Mannila1994] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo, Efficient Algorithms for Discovering Association Rules, *Proceedings of the AAAI Workshop on Knowledge Discovery in Databases (KDD-94),* pp. 181-192, July 1994.

[Meo1996] Rosa Meo, Giuseppe Psaila, and Stefano Ceri, A New Sql-Like Operator for Mining Association Rules, *Proceedings of the 22nd International Conference on Very Large Databases*, pp. 122-133, Mumbai, India, 1996.

[Miller1997] R. J. Miller and Yuping Yang, Association Rules over Interval Data, SIGMOD 1997, *Proceedings ACM SIGMOD International Conference on Management of Data*, May 13-15, 1997, Tucson, Arizona, USA, ACM Press, 1997, pp 452-461.

[Moore] Jerome Moore, Eui-Hong (Sam) Han, Daniel Boley, Maria Gini, Robert Gross, Kyle Hastings, George Karypis, Vipin Kumar, and Bamshad Mobasher, Web Page Categorization and Feature Selection Using Association Rule and Principle Component Clustering, *Dept. of Computer Science and Engineering, University of Minnesota, Minneapolis*, MN.

[Mueller1995] Andreas Mueller, Fast Sequential and Parallel Algorithms for Association Rule Mining: A Comparison, Technical Report CS-TR-3515, *Dept. of Computer Science, Univ. of Maryland, College Park, MD*, August 1995.

[Ng1998] Raymond T. Ng, Laks V.S. Laksmanan, Jiawei Han and Alex Pang, Exploratory Mining and Pruning Optimization of Constarinted Association Rules, *ACM Sigmoid, 1998*, Seattle, WA, USA. p. 13-24.

[Park1995] Jong Soo Park, Ming-Syan Chen, and Philip S. Yu, An Effective Hash Based Algorithm for Mining Association Rules, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pp. 175-186, San Jose, California, 22-25 May 1995.

[Park1995a] Jong Soo Park, Ming-Syan Chen, and PhilipS. Yu, Efficient Parallel Data Mining for Association Rules, *Proceedings of the International Conference on Information and Knowledge Management*, pp. 31-36, Baltimore, Maryland, 22-25 May 1995.

[Sarawagi1998], Sarawagi Sunita, Thomas Shiby, and Agrawal Rakesh, Integrating Mining with Relational Database Systems: Alternatives and Implications, Proceedings ACM SIGMOD International Conference on Management of Data, SIGMOD 1998, June 2-4, 1998, Seattle, Washington, USA

[Savasere1995] Ashoka Savasere, Edward Omiecinski, and Shamkant B. Navathe, An Efficient Algorithm for Mining Association Rules in Large Databases, *Proceedings of the 21nd International Conference on Very Large Databases*, pp. 432-444, Zurich, Swizerland, 1995.

[Shintani1996] Takahiko Shintani and Masaru Kitsuregawa, Hash Based Parallel Algorithms for Mining Association Rules, *Proceedings of PDIS*, 1996.

[Shintani1998] Takahiko Shintani and Masaru Kitsuregawa, Parallel Mining Algorithms for Generalized Association Rules with Classification Hierarchy, *Proceedings ACM SIGMOD International Conference on Management of Data, SIGMOD 1998,* June 2-4, 1998, Seattle, Washington, USA, pp 25-36.

[Silverstein1997] Craig Silverstein, Sergey Brin, and Rajeev Motwani, Beyond Market Baskets: Generalizing Association Rules to Dependence Rules, *Kluwer Academic Publishers*, Boston. Manufactured in Netherlands.

[Srikant1995] Ramakrishnan Srikant and Rakesh Agrawal, Mining Generalized Association Rules, *Proceedings of the 21nd International Conference on Very Large Databases*, pp. 407-419, Zurich, Swizerland, 1995.

[Srikant1996a] Ramakrishnan Srikant and Rakesh Agrawal, Mining Quantitative Association Rules in Large Relational Tables, *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pp. 1-12, Montreal, Quebec, Canada, 4-6 June 1996.

[Srikant1996b] Ramakrishnan Srikant, Fast Algorithms for Mining Association Rules and Sequential Patterns, *Ph.D Dissertation, 1996, University of Wisconsin*, Madison.

[Srikant1996c] Ramakrishnan Srikant, Quoc Vu and Rakesh Agrawal, Mining Association Rules with Item Constraints, *American Association for Artificial Intelligence*, 1997.

[Tank1998] Jian Tang, Using Incremental Pruning to Increase the Efficiency of Dynamic Itemset Counting for Mining Association Rules, *Proceedings of the 7th International Conference on Information and Knowledge Management*, pp. 273-280, Bethesda, Maryland, USA, 1998.

[Thomas1997] Shiby Thomas, Sreenath Bodagala, Khaled Alsabti and Sanjay Ranka, An Efficient Algorithm for the Incremental Updation of Association Rules in Large Databases, p. 263, *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD-97)*, 1997, p. 263.

[Toivonen1996] Hannu Toivonen, Sampling Large Databases for Association Rules, *Proceedings of the 22nd International Conference on Very Large Databases*, pp. 134-145, Mumbai, India, 1996.

[Tsur1998] Dick Tsur, Jeffrey D. Ullman, Serge Abiteboul, Chris Clifton, Rajeev Motwani, Svetlozar Nestorov, and Arnon Rosenthal, Query Flocks: a Generalization of Association-Rule Mining, *Proceedings ACM SIGMOD International Conference on Management of Data, SIGMOD 1998*, June 2-4, 1998, Seattle, Washington, USA.

[Tunk1999] Anthony K. H. Tung, Hongjun Lu, Jiawei Han and Ling Feng, Breaking the Barrier of Transactions: Mining Inter-Transaction Association Rules, *KDD 1999*, pp. 297-301.

[Xiao1999] Yongqiao Xiao and Margaret H Dunham, Considering Main Memory in Mining Association Rules, *Proceedings of DaWak*, 1999.

[Zaiane1998] Osmar R. Zaiane, Jiawei Han, Ze-Niam Li, and Joan Hou, Mining Mutimedia Data, *Intelligent Database System Research Laboratory, School of Computer Science, Simon Fraser University,* Burneaby, BC, Canada

[Zaki1996] Mohammed Javeed Zaki, Mitsunori Ogihara, Srinivasan Parthasarathy, and Wei Li, Parallel Data Mining for Association Rules on Shared-Memory Multiprocessors, *Technical Report TR 618, University of Rochester, Computer Science Department*, May 1996.

[Zaki1996a] Mohammed Javeed Zaki, Srinivasan Parthasarathy, Wei Li and Mitsunori Ogihara, Evaluation of Sampling for Data Mining of Association Rules, *University of Rochester, Computer Science Department TR 617*, May 1996

[Zaki1997] Mohammed Javeed Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li, New Parallel Algorithms for Fast Discovery of Association Rules, *Data Mining and Knowledge Discovery*, Vol. 1, No. 4, pp. 343-373, December 1997.

[Zaki1997a] Mohammed Javeed Zaki, Srinivasan Parthasarathy, and Wei Li, A Localized Algorithm for Parallel Association Mining, *Proceedings of the 9th ACM Symposium on Parallel Algorithms and Architectures,* 1997.

[Zaki1999] Mohammed Javeed Zaki, Parallel and Distributed Association Mining: A Survey, *IEEE Concurrency*, October-December 1999.

APPENDIX A: Sample Dataset

1) DatGen from Dataset Generator

DataGen is a computer code that creates data for user specified constraints. For example, data type can be sequential or random.
URL: http://www.datasetgenerator.com/source/
Language: C
Vendor: Data Generator
Cost: Free
Contact Info: melli@cs.sfu.ca; Phone/fax: (604) 291-3045

2) Quest Synthetic Data Generation Code

Quest synthetic Data Generation Code can be used to find large itemsets with/without taxonomies. It can also be used to obtain sequential patterns. There are two possible output formats for the data file: 1) Binary <CustID, TransID, NumItems, List-Of-Items> and 2) ASCII<CustID, TransID, and Item>
URL: http://www.almaden.ibm.com/cs/quest/demos.html
Functionality: Finding Large Itemsets as well as sequential pattern
Platform: UNIX
Language: C++
Cost: Free to download
Vendor: IBM Almaden Research Center
Contact Info: srikant@almaden.ibm.com

APPENDIX B: Sample Code

1) Magnum Opus for small datasets

Magnum Opus can be used to find association rules from a small dataset (up to 1000 entity) .
URL: http://www.rulequest.com/download.html
Functionality: Finding Association Rules
Platform: Windows 95/98/NT
Language: C

Cost: Free to download
Vendor: Rulequest Research
Contact Info: quinlan@rulequest.com ; Phone:+61 2 9449 6020; Fax: +61 2 9440 9272

APPENDIX C: Products

1) Intelligent Miner for Data from IBM

Intelligent Miner for Data from IBM can be used to identify and extract hidden information in data, uncovering associations, patterns, and trends through a process of knowledge discovery.

URL: http://www-4.ibm.com/software/data/iminer/fordata/index.html
Platforms: AIX, OS/390, OS/400, Solaris and Windows NT.
Vendor: IBM
Cost: 60,000 (approx)
Status: Commercial
Contact Info: E-mail: ibm_direct@vnet.ibm.com ;
Phone: 1-800-IBM-CALL
Fax: 1-800-2IBM-FAX

2) Intelligent Miner for Text from IBM

Intelligent Miner for Text can be used to extract necessary business information from a text data. It can also be used for searching text.

URL: http://www-4.ibm.com/software/data/iminer/fortext/index.html
Platforms: AIX, OS/390, Solaris and Windows NT
Vendor: IBM
Cost: Unknown
Status: Commercial
Contact Info: Visit Web site at www.software.ibm.com/data/iminer/fortext

3) Enterprise Miner from SAS

Enterprise Miner, an integrated software product for data mining mounted with graphical user interface (GUI), provides user to explore information from a massive database.

URL: www.sas.com
Platforms: AIX/6000, CMS, Compaq Tru64 UNIX, HP-UX, IRIX, Intel ABI, MVS, OS/2, OpenVMS Alpha, OpenVMS Vax, Solaris, Windows
Vendor: SAS Institute, Cary, NC, USA.
Cost: Not available.
Status: Commercially Developed
Contact Info**:** SAS Institute Inc. SAS Campus Drive Cary, NC 27513-2414, USA

4) MineSet from SGI

Using MineSet can be used to understand the complex patterns, relationships, and anomalies deeply hidden in a database.

URL: http://www.sgi.com/software/mineset
Platforms: Windows NT, Windows 98, Windows 95 and IRIX
Vendor: Silicon Graphics
Cost: Seat- US$995+, Server (Windows NT)- US$35,000+ and IRIX- US$50,000+
Status: Commercially Developed
Contact Info: Mark Olson , phone 415-933-2874, fax 415-932-2874
Email: mineset@sgi.com,
URL: http://www.sgi.com/software/mineset/contact.html

4) Clementine from SPSS

It consists of a number of tools to discover hidden rules in a user-friendly environment.

URL: www.spss.com/software/clementine
Platforms: Unix, WinNT
Vendor: SPSS Inc.
Cost: Non Known
Status: Commercially Developed
Contact Info: Phone: 1 (800) 521-1337, Fax: 1 (800) 841-0064

5) **Siftware: DBMiner**

DBMiner, an interactive mining of multiple-level knowledge in large relational databases, can be used to perform a wide variety of tasks. For example: generalization, characterization, association, classification, and prediction.

URL: http://db.cs.sfu.ca/DBMiner
Platform(s): Windows (95, NT), Unix
Status: Commercial
Cost: US$999.00
Contact**:** Jiawei Han, School of Computing Science
Simon Fraser University, Burnaby, B.C, Canada V5A 1S6
Telephone: (604)291-4411; Fax: (604)291-3045; Email: han@cs.sfu.ca

6) Classification Based on Association (CBA)

CBA is a data Mining tool developed at School of Computing, National University of Singapore. Its main algorithm was presented as a plenary paper "Integrating Classification and Association Rule Mining" in the 4th International Conference on Knowledge Discovery and Data Mining (KDD-98).

URL: http://www.comp.nus.edu.sg/~dm2/p_overview.html
Platforms: Windows 95, 98, NT 4.0
Vendor: National University of Singapore

Cost: Academic Version (Non-Commercial use): FREE; For commercial use: $1000.00
Status: Research Prototype
Contact Info: dm2@comp.nus.edu.sg

7) XpertRule Miner,

XpertRule Miner is a multipurpose data mining tool that supports the discovery of associations in both "case" and "transaction" data.

URL: http://www.attar.com/pages/info_xm.htm
Platforms: Microsoft Windows 95, 98 or NT
Vendor: Attar Software
Cost: US$ 4995
Status: Commercial
Contact Info:
In the USA**,** Canada and Mexico: Email: info@attar.com
Tel: 978 456 3946
Free call 800 456 3966
Fax: 978 456 8383
For the rest of the World: Email: info@attar.co.uk
Phone: 0870 606 0870 (inside the UK), +44 870 606 0870
Fax: 0870 604 0156 (inside the UK), +44 870 604 0156

8) **Magnum Opus (Association rule mining tools from RuleQuest) [Goebel1999]**

Magnum Opus can be used to find user specified maximum number of association rules from a substantial database containing both qualitative and numeric data.

URL: http://www.rulequest.com/MagnumOpus-info.html
latforms: Windows 95, 98, NT 4.0 or later
Vendor: Rulequest Research
Cost: US$740 (Single Computer)
Status: Commercial
Contact Info: Email: quinlan@rulequest.com
Phone: +61 2 9449 6020
Fax: +61 2 9440 9272

9) **Interestingness Analysis System (IAS) [Goebel1999]**

ISA can be used find interesting association rules after analyzing the discovered rules by allowing user's domain knowledge.

URL: http://www.comp.nus.edu.sg/~dm2/index.html
Vendor: Data Mining II, School of Computing, NUS, Singapore
Cost: Free for non-commercial use
Status: Research Prototype
Contact Info: dm2@comp.nus.edu.sg

APPENDIX D: Notation

Notation	Description
I	Set of items
I_d	Item (literal, attribute)
m	Number of items
D	Transaction database
S	Support
□	Confidence
T	Tuple in D
X,Y	Itemsets
$X \Rightarrow Y$	Association rule
L	Set of large itemsets
l	Large itemset
L_k	Set of large itemsets of size k
l_k	Large itemset of size k
C_k	Candidate sets of size k
$\overline{L_k}$	Set of large itemsets of size k and the TID containing them
$\overline{Ck}$	Set of candidate itemsets of size k and the TID containing them
D^i	Partition i for database D
X^i	Itemset for partition D^i
L^i	Set of large itemsets for partition D^i
C^i	Set of candidate itemsets for partition D^i
p	Number of partitions