International Journal of Computing and Corporate Research
ISSN (Online) : 2249-054X
Volume 4 Issue 2 March 2014
International Manuscript ID : 2249054XV4I2032014-16

# AN EMPIRICAL REVIEW ON ASSOCIATIVE RULE MINING AND CLASSIFICATION USING ASSORTED TECHNIQUES

*Anuradha*
*Research Scholar*
*Department of Computer Science and Applications*
*Kurukshetra University, Kurukshetra*

*Dr. Chanderkant Verma*
*Assistant Professor*
*Department of Computer Science and Applications*
*Kurukshetra University, Kurukshetra*

**ABSTRACT**

Association rule mining and standards, initially presented in 1993, are utilized to distinguish connections around a set of things in a database. These connections are not dependent upon innate properties of the information themselves (as with practical conditions), but instead dependent upon co-event of the information things. Association principles are additionally utilized for different requisitions, for example, expectation of disappointment in telecommunications arranges by distinguishing what occasions happen before a disappointment. The vast majority of our attention in this paper will be on bushel market dissection, however in later segments we will take a gander at different provisions too. Acquaintanceship guidelines are a standout amongst the most investigated zones of information mining and have as of late accepted much consideration from the database group. They have ended up being very convenient in the promoting and retail groups and other more various fields. In this paper we give a review of acquaintanceship guideline research. The target of this paper is to give a careful review of past exploration on affiliation principles.

**Keywords -** Association Rule Mining, Data Mining, Machine Learning, Associative Rule Mining

International Journal of Computing and Corporate Research
ISSN (Online) : 2249-054X
Volume 4 Issue 2 March 2014
International Manuscript ID : 2249054XV4I2032014-16

**INTRODUCTION**

Data mining is a powerful situated of dissection apparatuses and systems utilized within the decision support process. Notwithstanding, misguided judgments about the part that data mining plays in choice help results can prompt perplexity about and abuse of these apparatuses and methods. Data Mining is the disclosure of concealed data found in databases and could be seen as a venture in the learning finding methodology. Information mining capacities incorporate bunching, grouping, forecast, join investigation and cooperations. One of the most paramount information mining requisitions is that of mining acquaintanceship guidelines.

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 $X \bigcap Y = \phi$. Here, X is called antecedent, and Y consequent. Two important measures for association rules, support (s) and confidence ($\alpha$), can be defined as follows.

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

International Journal of Computing and Corporate Research
ISSN (Online) : 2249-054X
Volume 4 Issue 2 March 2014
International Manuscript ID : 2249054XV4I2032014-16

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, $\alpha$
**Output**:
Association rules satisfying s and $\alpha$
**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, $\alpha$.

**Algorithm 2. Find Association Rules Given Large Itemsets**:
**Input**:
I, D, s, $\alpha$, L
**Output**:
Association rules satisfying s and $\alpha$
**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.

## CRUCIAL ALGORITHMS

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.

### Sequential Algorithms

#### 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 $\alpha$ is the confidence of the rule. The AIS algorithm makes multiple passes over

International Journal of Computing and Corporate Research
ISSN (Online) : 2249-054X
Volume 4 Issue 2 March 2014
International Manuscript ID : 2249054XV4I2032014-16

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.

**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].

International Journal of Computing and Corporate Research
ISSN (Online) : 2249-054X
Volume 4 Issue 2 March 2014
International Manuscript ID : 2249054XV4I2032014-16

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.

**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.

**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

International Journal of Computing and Corporate Research
ISSN (Online) : 2249-054X
Volume 4 Issue 2 March 2014
International Manuscript ID : 2249054XV4I2032014-16

transaction. It should be mentioned that each entry in $\overline{C_k}$ may be larger than the corresponding transaction for smaller k values [Srikant1996b].

**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].

**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.

**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.

International Journal of Computing and Corporate Research
ISSN (Online) : 2249-054X
Volume 4 Issue 2 March 2014
International Manuscript ID : 2249054XV4I2032014-16

**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 $BD^{-}(PL) \bigcup PL$. 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 $BD^{-}(PL) \bigcup PL$ 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.

**Algorithm : 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

International Journal of Computing and Corporate Research
ISSN (Online) : 2249-054X
Volume 4 Issue 2 March 2014
International Manuscript ID : 2249054XV4I2032014-16

12) count(MC, D);
13) return $L = \{x \mid x \in C \bigcup MC, x.count \geq s \times |D|\}$;

**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.

**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.

**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

International Journal of Computing and Corporate Research
ISSN (Online) : 2249-054X
Volume 4 Issue 2 March 2014
International Manuscript ID : 2249054XV4I2032014-16

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).

**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.

**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 : 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

International Journal of Computing and Corporate Research
ISSN (Online) : 2249-054X
Volume 4 Issue 2 March 2014
International Manuscript ID : 2249054XV4I2032014-16

```
        //to obtain the global counts in the whole database.
4)      forall itemset X ∈ C_k do begin
5)          X.count=∑_{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 ∈ C_k | c.count ≥ s × | D^1∪D^2∪…∪D^p |};
8)      C_{k+1}=apriori_gen(L_k);
9)  end
10) return L=L_1 ⋃ L_2 ⋃ … ⋃ L_k;
```

**PDM** (Parallel Data Mining)

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.

**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].

International Journal of Computing and Corporate Research
ISSN (Online) : 2249-054X
Volume 4 Issue 2 March 2014
International Manuscript ID : 2249054XV4I2032014-16

**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.

**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.

**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 : DD [Agrawal 1996]**
**Input:**
$I, s, D^1, D^2, \ldots, D^p$
**Output:**
L
**Algorithm:**
1) $C_1^i \subseteq I$;
2) **for** $(k=1; C_k^i \neq \emptyset; k++)$ **do begin**

International Journal of Computing and Corporate Research
ISSN (Online) : 2249-054X
Volume 4 Issue 2 March 2014
International Manuscript ID : 2249054XV4I2032014-16

```
                //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≤j≤p, j≠i) to get the global counts.
4)      broadcast(D^i);
5)      for (j=1; (j≤p and j≠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∈C^i_k, c.count ≥ s*|D^1∪D^2∪…∪D^p|};
10)     L_k= ⋃_{i=1}^{p}(L_k^i);
11)     C_{k+1} = apriori_gen(L_k);
12)     C_{k+1}^i ⊆ C_{k+1}; //partition the candidate itemsets across the processors
13) end
14)  return L = L_1 ⋃ L_2 ⋃ … ⋃ L_k;
```

**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.

**Comparison of Algorithms**

| Algorithm | Scan | Comments |
|---|---|---|
| AIS | m+1 | Suitable for low cardinality sparse transaction database; Single consequent |
| SETM | m+1 | SQL compatible |

International Journal of Computing and Corporate Research
ISSN (Online) : 2249-054X
Volume 4 Issue 2 March 2014
International Manuscript ID : 2249054XV4I2032014-16

| | | |
|---|---|---|
| Apriori | m+1 | Transaction database with moderate cardinality; Outperforms both AIS and SETM; Base algorithm for parallel algorithms |
| Apriori-TID | m+1 | Very slow with larger number of $\overline{C_k}$; Outperforms Apriori with smaller number of $\overline{C_k}$; |
| Apriori-Hybrid | m+1 | Better than Apriori. However, switching from Apriori to Apriori-TID is expensive; Very crucial to figure out the transition point. |
| OCD | 2 | Applicable in large DB with lower support threshold. |
| Partition | 2 | Suitable for large DB with high cardinality of data; Favors homogenous data distribution |
| Sampling | 2 | Applicable in very large DB with lower support. |
| DIC | Depends on interval size | Database viewed as intervals of transactions; Candidates of increased size are generated at the end of an interval |
| CARMA | 2 | 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 | Data Parallelism. |
| PDM | m+1 | Data Parallelism; with early candidate pruning |
| DMA | m+1 | Data Parallelism; with candidate pruning |

| **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 |

International Journal of Computing and Corporate Research
ISSN (Online) : 2249-054X
Volume 4 Issue 2 March 2014
International Manuscript ID : 2249054XV4I2032014-16

## CONCLUSION AND SUMMARY

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). The problem with incremental updating is to find the large itemsets for a database D∪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. Another approach to maintaining association rules is based on the idea of sampling.

## REFERENCES

[1] [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.

[2] [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.

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

[4] [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.

[5] [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.

[6] [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.

[7] 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.

International Journal of Computing and Corporate Research
ISSN (Online) : 2249-054X
Volume 4 Issue 2 March 2014
International Manuscript ID : 2249054XV4I2032014-16

[8] 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.

[9] [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.

[10] 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

[11] [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.

[12] [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.

[13] [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.

[14] [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.

[15] [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.

[16] [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.

[17] [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.

[18] [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.

[19] [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.

[20] [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.