CHAPTER 5

WEIGHTED SUPPORT ASSOCIATION RULE MINING
USING CLOSED ITEMSET LATTICES IN PARALLEL

5.1 INTRODUCTION

During recent years, one of the vibrant research topics is Association rule discovery. This was first introduced in Agarwal et al (1993). The association rule discovery is used to identify relationships between items in very large databases, to extract interesting correlations and associations among sets of items in the transaction databases or other data repositories. For example, given a market basket database, it would be interesting for decision support to know that 30% of customers who bought coca powder and sugar also bought butter. This analysis may be used to provide some basis if it is required to increase the sales and introduce free schemes like, if 3 kg of sugar is bought then 100g butter free is given. In a census database, it is discovered that 20% of persons who worked last year earned more than the average income, or in a medical database, that 35% of patients who have cold also have sinus (Pasquier et al 1999). There are many areas in which association rules are widely used. Among them may be listed out some of areas such as telecommunication networks, market and risk management, inventory control etc. After the publication of the research papers of Agarwal et al (1993) and Agarwal and Srikant (1994), discovering association rules, for the given threshold of minimum support and confidence, has become one of the most active research topics.
There are two sub-problems in association rule mining. Finding frequent or large itemsets is the first sub-problem. Frequent itemsets are those itemsets whose frequency of occurrence or support is greater than the minimum support provided by the user in the database. Generating association rules from the frequent itemsets generated in the first step is the second problem. Association rules must satisfy the minimal confidence constraint. Suppose one of the large itemsets is \( L_k \), \( L_k = \{i_1, i_2, \ldots, i_k\} \), this itemset can be used to generate the association rules like the following: the first rule is \( \{i_1, i_2, \ldots, i_{k-1}\} \rightarrow \{i_k\} \), this rule can be determined whether it is interesting or not by using the confidence constraint (Sotiris Kotsiantis and Dimitris Kanellopoulos 2006). The different rules are generated from this rule itself by deleting the last items on the left side of the rule and appending it to the right of the rule. The confidence constraint is used now to determine the corresponding rule’s interesting. This process continues till all the items come to the right of the rule. The second sub-problem is simpler when compared to the first sub-problem. Once the frequent items are found, it is a straightforward procedure to generate association rules with the provided support. Hence, the problem of mining association rules is reduced to the problem of finding frequent itemsets.

Frequent items can be generated in two steps. Firstly, candidate large itemsets are generated and secondly frequent itemsets are generated using these candidate itemsets. The itemsets whose support is greater than the minimum support are referred as frequent itemsets. The itemsets that are expected to be large or frequent are termed as candidate itemsets. The drawback in validating the large number of association rules that are generated limits the applications of data mining. There is very vast literature survey done to reduce the number of association rules. Some of these algorithms stated that the association rules can be generated without duplicates or only interesting rules based on strength (Sotiris Kotsiantis and
Dimitris Kanellopoulos 2006). Mostly all algorithms that are proposed to generate association rules are based on Apriori mining method (Agarwal and Srikant 1994). The performance of such algorithms is good for weakly correlated data as market basket data but is bad for correlated data such as census data. The significance of the attributes in a transaction within the whole item space is considered to be same without its significance in traditional association rule mining. If the association rules are generated in this fashion, some interesting rules are missed. For example, \([\text{wine} \rightarrow \text{salmon}, 1\%, 80\%]\) may be more important than \([\text{bread} \rightarrow \text{milk}, 3\%, 80\%]\) even though the former holds a lower support (Feng Tao et al 2003). This is because those items in the first rule usually come with more profit per unit sale, but the standard ARM simply ignores this difference.

The model in (Wang et al 2000) also considers only whether an item is present in a transaction, but does not take into account the weight/intensity of an item within a transaction. For example, a customer may purchase 13 bottles of coke and 6 bags of snacks and another may purchase 4 bottles of coke and 1 bag of snacks at a time. The conventional association rule approach treats the above two transactions in the same manner, which could lead to the loss of some vital information. Assume, for example, that if a customer buys more than 7 bottles of coke, he is likely to purchase 3 or more bags of snacks. Otherwise, the purchase tendency of coke is not strong. The traditional association rule cannot express this type of relationship. With this knowledge, the supermarket manager may set a promotion strategy, such as if a customer buys 8 bottles of coke; he can get two free bags of snacks. So, weight is associated for the items through which rules are found. Weighted Association Rules cannot only improve the confidence in the rules, but also provide a mechanism to do more effective target marketing by identifying or segmenting customers based on their potential degree of loyalty or volume of purchases (Wang et al 2000). The main challenge of adapting traditional
association rule mining model in a weighted setting is the invalidation of the “downward closure property”, which is used to justify the efficient iterative process of generating and pruning large itemsets from its subsets.

5.2 BACKGROUND

The basic concepts of association rule mining and its preliminaries are discussed by Sotiris Kotsiantis and Dimitris Kanellopoulos (2006). They also conducted a survey of the existing association rule mining techniques. The new and efficient algorithm, Close is proposed by Pasquier et al (1999). It is based on pruning the closed set lattice. Closed itemset lattice is a sub-order of the subset lattice and is closely related to Wille’s concept lattice in formal concept analysis (Wille 1982). Traditional association rule problem is extended by Wang et al (2000). In this algorithm, the intensity of the item in the transaction is considered and a weight attribute is associated with each item based on its intensity. The rule generated from the items associated with weight is referred as weighted association rule (WAR). They also discussed how the confidence of the rules can be improved and effective target marketing can be achieved if the customers are divided based on their potential degree of loyalty or the volume they purchase. In WAR, the frequent itemsets are found by ignoring the weight and the weight is associated during the generation of association rules.

Agarwal and Srikant (1994) presented two new algorithms for solving the problem of discovering association rules between items in a large database of sales transactions, which are fundamentally different from the known algorithms. Empirical evaluation shows that these algorithms outperform the known algorithms by factors ranging from three for small problems to more than an order of magnitude for large problems. The best features of the two proposed algorithms can be combined into a hybrid algorithm, called AprioriHybrid. Feng Tao et al (2003) addressed the issues of
discovering significant binary relationships in transaction datasets in a weighted setting. Traditional model of association rule mining is adapted to handle weighted association rule mining problems where each item is allowed to have a weight. The focus is on those significant relationships involving items with significant weights rather than allowing them to be flooded in the combinatorial explosion of insignificant relationships. The challenge in this approach is using weights in the iterative process of generating large itemsets. The problem of invalidation of the “downward closure property” in the weighted setting is solved by using an improved model of weighted support measurements and exploiting a “weighted downward closure property”. A new algorithm called WARM (Weighted Association Rule Mining) is developed based on the improved model. The algorithm is both scalable and efficient in discovering significant relationships in weighted settings as illustrated by experiments performed on simulated datasets.

Agarwal and Shafer (1996) presented three algorithms and explored tradeoffs between computation, communication, memory usage and synchronization. The three algorithms are count distribution, data distribution and candidate distribution. Count distribution algorithm is what motivated the proposed work. In count distribution algorithm, Agarwal considers only the support of the item to generate frequent itemsets. It became imperative to consider the significance of the item also and to generate closed frequent itemsets. These are the two variations in the proposed work from the count distribution algorithm. By considering the significance of the item, important items are not lost. Their support may be less than the minimum support provided by the user because of its durability. By generating closed frequent itemsets, the number of itemsets in each level reduces, so that the time taken will be reduced. Moreover any interesting or significant itemsets will not be lost as all frequent itemset can be generated from closed frequent itemsets.
5.3 PROPOSED WORK

In general, the association rules are generated in two steps. First, frequent itemsets are found and secondly, rules are generated using the frequent itemsets found in first step. Frequent itemsets are also very huge in order to perform any analysis or to generate association rules. Instead, closed frequent itemsets are generated from which association rules can be formed. Generally, in generating closed frequent itemsets, minimum support is only considered. But if minimum support alone is considered, some interesting / important items whose support < minimum support are lost. So, consider a special attribute referred as weight which is associated with each item and it has a value based on its durability / expiry / significance. For example, the lifetime of bread or cheese is less when compared to the lifetime of wheat, rice etc in market basket database. The quantity of purchase also can be considered as a weight. To improve the performance, parallel algorithm count distribution is used to generate closed frequent itemsets and rules are also generated in parallel. Count distribution algorithm focuses on minimizing communication. It does so at the expense redundant computations being carried out in parallel. The principle of allowing “redundant computations in parallel on otherwise idle processors to avoid communication” is followed in this algorithm.

5.3.1 Modified Count Distribution Algorithm

5.3.1.1 Generating Closed Frequent itemsets

1. Minimum weighted support is provided by the user.

2. Weight is associated with each item in the transaction database.

3. The database is distributed among ‘N’ different processors.
4. In first pass, each processor scans its local database and generates local candidate itemset.

5. Support of each item is found based on its frequency in the local database of a particular processor.

6. Weighted support of each item in the database is calculated as weight * support.

7. The weighted supports (count) of all items are normalized to fix in some defined range of values.

8. The local counts are communicated among all the processors to develop global counts.
   
a. Local candidate set of each processor $P_i$ is maintained in a closed hash-table. For each tuple, every item is hashed and its corresponding count in the hash table is incremented. New entries are created if necessary.

b. At the end of the pass, processor loads items and their counts from the hash table into a send buffer $\text{ItemsOfProc}_i$ and then gathers items and their support counts from all other processors.

c. To do this, it must first gather the count of the total number of items residing in the send buffers of all other processors.

d. Processor $P_i$ puts the count of its own items in a $\text{CountBuf}$ and calls $\text{AllGather}(\text{SendBuf} = \text{CountBuf}, \text{ReceiveBuf} = \text{CountArr}, \text{BlockLen} = \text{sizeof( integer)}).$ The $j^\text{th}$ element of the $\text{CountArr}$ now contains the number of items processor $j$ has in its send buffer.
e. Next, processor $P_i$ calls $\text{AllGatherV}(\ )$ to collect all items and their counts into the receive buffer $\text{AllItems}$

f. $\text{AllGatherV}(\text{SendBuf} = \text{ItemsOfProcI}, \text{ReceiveBuf} = \text{AllItems},\text{BlockLen} = \text{sizeof(ItemsOfProcI)}, \text{ReceiveBlockLen} = \text{CountArr})$

g. $\text{AllGatherV}()$ is the variable length counterpart of $\text{AllGather}()$ in which a processor receives messages of different sizes from other processors. $\text{SendBuf}$ is of size $\text{BlockLen}$, $\text{ReceiveBuf}$ is an array of $N$ messages, and the size of the $i^{th}$ receive buffer is given by the $i^{th}$ element of the $\text{ReceiveBlockLen}$ array. If $\text{AllItems}$ array becomes too large, an intermediate step uses $\text{ReduceScatter}()$ to reduce the number of duplicate entries. This detail is omitted for brevity.

h. $P_i$ now hashes items from the receive buffer into a new hash table. If the same item was counted by more than one processor, it will hash to the same bucket and the support count for this item is accumulated. Thus, $P_i$ now has the entire candidate set $C_1$, complete with global counts.

9. These global counts are used to generate frequent itemsets $L_k$.

10. Closed frequent itemsets $CL_k$ are found from these frequent itemsets by removing every frequent itemset that is a proper subset of, and carries the same support as, an existing frequent itemset.

11. When $k > 1$ then
12. Each processor $P_i$ generates the complete candidate set $C_k$ using the closed frequent itemset $CL_{k-1}$ created at the end of pass $k - 1$. Now the candidate set is identical in all the processors because of identical $CL_{k-1}$.

13. Local counts (weighted supports) for the candidate set are developed by making a pass over its local database.

14. Weighted Support of an itemset is calculated as the average weight of all the items in the set (sum of weights of all items/number of items in the set).

15. All processors exchange local counts with the other to generate global counts. Here synchronization of processors is forced.

   a. The candidates are kept in a hash-tree to allow efficient counting when making a pass over the data.

   b. To exchange local counts, each processor asynchronously extracts its local counts of candidate sets into a count array $LCntArr$.

   c. Since candidate set $C_k$ is identical for all processors, if every processor traverses $C_k$ in exactly the same order then corresponding elements of the count arrays will correspond to identical candidate itemsets.

   d. Thus it is not necessary to communicate itemsets themselves but only their counts. It also saves on computation because these local counts can be summed using simple vector summation rather than having to compare and match candidates.
e. Having created LCntArr, processors now, do ReduceScatter() communication to perform a partitioned vector-sum of the count arrays.

ReduceScatter(SendBuff=LCntArr,ReceiveBuf=PartGCntArr, BlockLen= PartSize, ReductionFunction=add)

f. As the result of this operation, processor $P_i$ receives the global counts of all the items in the $i$th LCntArr partition of all the processors.

g. The number of items in each partition, PartSize, will be sizeof(LCntArr)/N.

h. Each processor now gathers into GCntArr the global counts of items belonging to all other partitions by calling AllGather(SendBuf=PartGCnt,ReceiveBuf=GCntArr,BlockLen=PartSize).

i. Thus giving each processor the global counts for all candidates in $C_k$.

16. Each processor $P_i$ now computes $L_k$ from $C_k$.

17. Then closed frequent itemset $CL_k$ is generated from $L_k$ by each processor.

18. The decision to terminate or continue the next pass will be taken independently by each processor. The decision will be identical as the processors all have identical frequent itemsets.

Thus the processors scan their local data asynchronously in parallel in every pass. At the end of each pass, they are synchronized to develop global counts.
5.3.1.2 Association Rule Generation in Parallel

Now it is time to present a parallel implementation of the second subproblem – the problem of generating rules from closed frequent itemsets. Generating rules is much less expensive than discovering closed frequent itemsets as it does not require examination of the data. Given a closed frequent itemset $L$, rule generation examines each non-empty subset `$a` and generates the rule $a \Rightarrow (L - a)$ with support = support ($L$) and confidence = support ($L$)/support ($a$). This computation can efficiently be done by examining the largest subsets of $L$ first and only by proceeding to smaller subsets if the generated rules have the required minimum confidence (Savasere et al 1995). For example, given a closed frequent itemset $ABCD$, if the rule $ABC \Rightarrow D$ does not have minimum confidence, neither will $AB \Rightarrow CD$, and so there is need to consider it.

Generating rules in parallel simply involves partitioning the set of all closed frequent itemsets among the processors. Each processor then generates rules for its partition only by using the aforesaid algorithm. Since the number of rules that can be generated from an itemset is sensitive to the itemsets’ size, equitable balancing is attempted by partitioning the itemsets of each length equally across the processors. Note that in the calculation of the confidence of a rule, a processor may need to examine the weighted support of an itemset for which it is not responsible. For this reason, each processor must have access to all the closed frequent itemsets before rule generation can begin. This is not a problem because all the processors have all the closed frequent itemsets at the end of the last pass.
5.4 PERFORMANCE EVALUATION

In performance evaluation, synthetic datasets of varying complexity are used for experiments. The datasets on which the experiments are carried out is D1456K.T15.I4. Experiments were repeated many times to obtain stable values at each data point. The characteristics like scaleup, sizeup and speedup are examined and compared with existing count distribution algorithm. The results clearly indicate that the proposed algorithm performs better when compared to the count distribution algorithm.

Figure 5.1 Comparative performance of Count Distribution and the proposed algorithm in terms of absolute response time with varied number of processors and varied database sizes
Scaleup experiments were done in order to test how the proposed algorithm handled the larger datasets when more processors are available. The results shown in Figure 5.1 indicate clearly that the proposed algorithm performs considerably well by maintaining response time almost constant as the database and the number of processors increase. It has also been shown that the results in terms of scaleup which is the response time normalized with respect to the response time for a single processor. Figure 5.2 displays it

![Relative Scaleup](image)

**Figure 5.2** Comparative performance of Count Distribution and the proposed algorithm in terms of Scaleup with varied number of processors and varied database sizes
Figure 5.3  Comparative performance of Count Distribution and the proposed algorithm in terms of absolute response time with varied database sizes and 16 nodes

The experiments in terms of sizeup are done by fixing the size of the multiprocessor at 32 nodes while growing the database from 25MB per node to 400 MB per node and results are shown in Figure 5.3. The sizeup is the response time normalized with respect to the response time for 25MB per node. The results show that algorithm performs better as the database size is increased. The performance in terms of sizeup is depicted in Figure 5.4.
Figure 5.4 Comparative performance of Count Distribution and the proposed algorithm in terms of sizeup with 16 processors and varied database sizes
The speedup experiments are done by keeping the database as constant and by varying the number of processors. The results are shown in Figure 5.5. The database size is fixed at 400MB. The speed up is the response time normalized with respect to the response time for a single processor and the performance is shown in Figure 5.6. It may be noticed from the experiments that the more data a node processes, the less significant becomes the communication time thereby giving us a better performance.
Figure 5.6  Comparative performance of Count Distribution and the proposed algorithm in terms of speedup with varied number of processors and constant database size of 400MB

5.5  SUMMARY

As there are wide applications of association rules in data mining, it is important to provide good performance. In view of this, a new algorithm is proposed which considers not only the significance of the item but also the support of the item. The database is partitioned across different processors to generate the frequent itemsets in parallel. Moreover, concentration has been given to generate closed frequent itemsets instead of frequent itemsets because the number of closed frequent itemsets will be smaller than the number of frequent itemsets. As all the frequent itemsets can be obtained from the closed frequent itemsets, it is enough to find the closed frequent itemsets. Similar to the count distribution algorithm, the proposed algorithm
exchanges only weighted supports of the items among the processors and minimizes communication. Subsequently, the performance of the proposed algorithm has been compared with the existing count distribution algorithm in terms of scaleup, sizeup and speedup. The results have shown that the proposed algorithm exhibits better performance.