An Improved Algorithm for Mining Association Rules Using Multiple Support Values

Ioannis N. Kouris, Christos H. Makris, Athanasios K. Tsakalidis

University of Patras, School of Engineering
Department of Computer Engineering and Informatics, 26500 Patras, Hellas (Greece)
&
Computer Technology Institute, P.O. BOX 1192, 26110 Patras, Hellas (Greece)
E-mails: jkouris@ceid.upatras.gr, makri@ceid.upatras.gr, tsak@ceid.upatras.gr

Abstract

Almost all the approaches in association rule mining suggested the use of a single minimum support, technique that either rules out all infrequent itemsets or suffers from the bottleneck of generating and examining too many candidate large itemsets. In this paper we consider the combination of two well-known algorithms, namely algorithm DIC and MSApriori in order to end up with a more effective and fast solution for mining association rules among items, with different support values. Experiments show that the new algorithm is better than algorithm MSApriori, as well as better than algorithm DIC.

Introduction

Since its introduction from Agrawal, Imielinski and Swani (1993), the task of association rule mining has received much attention, and still remains one of the most popular pattern discovery methods in KDD. A formal description of the problem can be found from (Agrawal, Imielinski and Swani 1993), or from (Agrawal and Srikant 1994). Many alternatives have been proposed in order to improve this first approach (Toivonen 1996; Park, Chen and Yu 1995; Brin et. al. 1997), with most of the subsequent algorithms trying to reduce the I/O overhead. Lately much effort has been made in the on-line generation of association rules (Aggarwal and Yu 2001; Hidler 1997; Kouris, Makris and Tsakalidis 2002). For a survey on the bulk of the work made on the area of data mining readers are referred to the work of Chen, Han and Yu (n.d.). However almost all the approaches suggest the use of a single minimum support, assuming all itemsets to be of the same importance.

Mannila (1998) presented what he called the rare item problem. According to this problem if the overall minsup is set too high, then we eliminate all those itemsets that are infrequent. On the other hand if we set the overall minsup too low, in order to find also those itemsets that are infrequent, we will most certainly generate a huge number of frequent itemsets and will produce far too many rules, rules that will be meaningless. In order to deal with this problem there have been proposed various alternatives. One is splitting the data into a few blocks according to the frequencies of the items and then mine for rules in each distinct block with a different minimum support (Lee, Stolfo and Mok 1998). One second approach is to group a number of related rare items together into a higher order item, so that this higher order item is more frequent (Lee, Stolfo and Mok 1998; Han and Fu 1995; Srikant and Agrawal 1995). However the first approach fails to produce in a straightforward - simple manner rules with items across different blocks, while the second approach fails to produce rules involving the individual rare items that form the higher order itemset.

Liu, Hsu and Ma (1999) proposed an algorithm called MSApriori, which was based on algorithm Apriori and could find rules among items with different supports without falling in the pitfall described by Mannila (1998). According to this approach, every itemset in the database can have its own support value called MIS. Let MIS(i) denote the MIS value of an item i. The minimum support of a rule R is the lowest MIS value among the items in the rule. That is, a rule R involving k items is valid if the rule's actual support is greater or equal to:

$$[\min(MIS(a_1),MIS(a_2),\ldots,MIS(a_k))]$$

That way we are in position to have higher minimum supports for rules only with frequent itemsets and lower minimum supports for rules that involve also rare itemsets. However the use of multiple supports takes off the advantage of pruning and discarding some itemsets from further consideration. All level-wise algorithms like Apriori, rely on the property that an itemset can only be large if only all of its subsets are large (downward closure property) so that they can reduce the number of itemsets that have to be checked. In order not to miss that advantage the specific researchers suggested a variation called the *sorted closure property*, according to which all the 1-items in the dataset are sorted in ascending order according to their MIS values. This ordering is used in all subsequent operations, and the items in all higher order itemsets follow this ordering. Nevertheless MSApriori, like its ancestor Apriori, required as many passes over the data as is the number of single items in the higher order candidate itemset.

On the other hand Brin et al. (1997) presented algorithm DIC, which made use of a single minimum support and reduced the number of passes made over the data while at the same time kept the number of itemsets that have to be checked low as compared to sampling techniques (Toivonen 1996). The main idea of this algorithm was to start counting any itemset as soon as it is suspected to be large instead of waiting until all of its subsets are counted through all the transactions. According to Brin, DIC is compared to a train running over the data with stops at intervals M transactions apart. At every stop of the train some new itemsets get on the train (start to be counted), while others get of (stop to be counted). What this approach fails to approximate is that not all passengers are the same. Some passengers are first class passengers and thus need some extra attention, while some others are the economy class. Every itemset that gets on the train at any stop is presumed to be the same as all the others, or in other words to have the same support threshold as all the others. What we try to manage is to incorporate the philosophy of the multiple support values introduced by Liu, Hsu and Ma (1999), into the procedure proposed by Brin et al. (1997) and end up with an even faster and more efficient algorithm for mining association rules with multiple minimum supports.

Algorithm KTM

Every itemset in our database is marked with a different state. Apart from the four states that were used also by the DIC algorithm, we introduce two additional states for our purposes. So our algorithm marks overall the itemsets in six different possible ways, which are:

Dashed Circle (DC) – suspected small itemset – an itemset we are still counting that its count is bellow its MIS value– also the initial state of all itemsets.

Solid Circle (SC) – confirmed small itemset – an itemset we have finished counting and its count is bellow its own MIS value as well as bellow the MIS value of the item before it.

Dashed Square (DS) – suspected large itemset – an itemset we are still counting but its count already exceeds its MIS value.

Solid Square (SS) – confirmed large itemset – an itemset we have finished counting through all the transactions and that exceeds its MIS value.

Dashed Diamond (DD) – itemset which we confirmed that its count is bellow its own MIS value but above the MIS value of the item before it in the MIS ordering, and has not yet generated its corresponding candidate itemsets.

Solid Diamond (SD) – itemset that we confirmed that its count is bellow its own MIS value but above the MIS value of the item before it in the MIS ordering and that has generated its corresponding candidate itemsets.

Every 1-itemset begins to be counted with its state DC – Dashed Circle, except from the empty itemset, which is marked immediately with its state solid box (Figure 1, snap 1). After M transactions, where M is the number of transactions that have to be read before our train (algorithm) makes its first stop, we check the counter of every itemset we are counting against its MIS value. If its counter is larger or equal to its MIS value then its state is changed into DS – Dashed Square (Figure 1, snap 2).

When an itemset has been counted through all the transactions we check again its counter against its MIS value. If its state was DC – Dashed Circle, and its counter is finally equal or larger than its MIS value we change its state to SS – Solid Square. If its state was DS – Dashed Square, we simply change it to SS – Solid Square (Figure 1, snap 3).

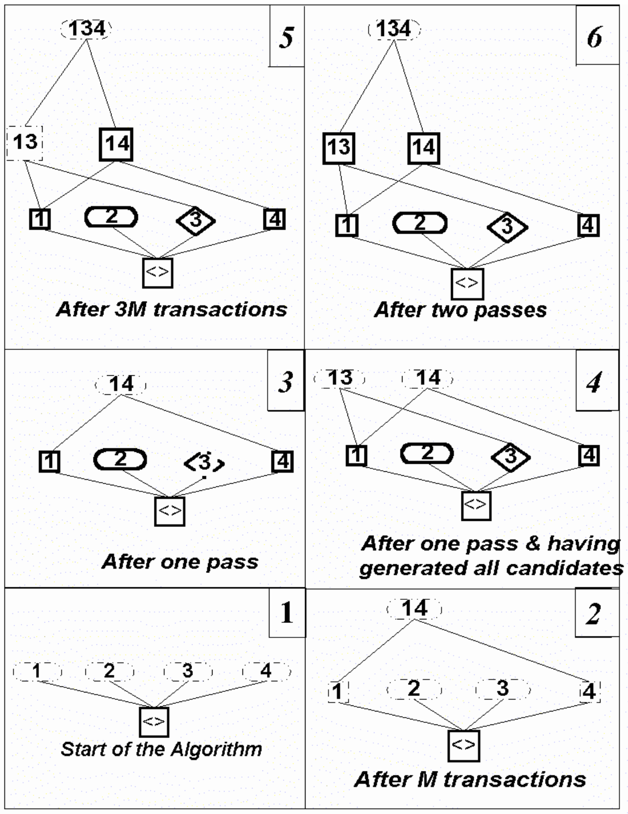


Figure 1: The running of the KTM algorithm

If its counter remains lower than its MIS value and it is a 1-itemset we now check it against the MIS value of the item before it in the MIS ordering. If it is also bellow the MIS value of the item before it, we then change its state to SC – Solid Circle. If on the other hand its counter is equal or over the MIS value of the item before it, we change its state to DD – Dashed Diamond (Figure 1, snap 3), and begin generating all candidates from it. This state is not preserved long since after this itemset has generated all possible candidate itemsets we change its state to SD – Solid Diamond (Figure 1 snap 4). The only difference between the two states is that an itemset with state SD has generated all possible candidate itemsets from it. The reason we added two states, Dashed Diamond - DD and Solid Diamond – SD instead of just one (state DD – Dashed Diamond) was because we wanted to keep our algorithm from producing new candidates at every stop.

Our algorithm terminates upon the absence of any dashed itemset.

If any immediate superset of a 1-itemset stated DS – Dashed Square, has all of its subsets as solid or dashed squares, we make its state DC – Dashed Circle and begin counting it (Figure 1 snaps 2 & 5). This procedure is somewhat different for 1-itemsets which are stated DD – Dashed Diamonds in that we generate supersets from them by using the 1-itemset stated DD and any 1-itemset stated DS – Dashed Square, or SD – Solid Square which is also before it in the MIS ordering (Figure 1 snap 4 where we generated the candidate itemset <1,3> only and not also <3,4>). Also the order of every new itemset is preserved, by putting first the 1-itemset that has the lowest MIS value, and then all the other items sorted ascending by their MIS values.

If any immediate superset of a k-itemset ($k \geq 2$) stated DS – Dashed Square has all of its subsets as solid or dashed squares, we make its state DC – Dashed Circle and begin counting it. There is however one exception when a subset of an itemset is not large (i.e. is not stated DS or SS), and nevertheless the candidate superset cannot be pruned (In Figure 1, snap 4 superset <1,3,4> is generated although subset <3,4> is not large). This case arises when the subset does not contain the first item of the superset (there is always one such subset) and we are not sure that the MIS value of the first item is not the same as that of the second item.

As we can easily understand only 1-itemsets can be stated DD or SD, since we keep an MIS ordering only for those itemsets. All the other higher order itemsets can be of state DC, DS, SS and SC.

Upon termination of the algorithm we end up with an itemsets lattice like the one shown in Figure 2, with the empty itemset at the bottom and the set of all items at the top. The itemsets that are not marked with any state at all (i.e. were never counted) are nevertheless shown here for the ease of representation.

Item ordering

The ordering of the items in the data structure plays an essential role in both algorithms, each for its own purposes. In this section we show that the ordering that we apply serves both algorithms and results in a more efficient combined algorithm.

Liu, Hsu and Ma (1999) proposed a sorting of the items according to their MIS values in ascending order in order to satisfy what they called the sorted closure property. Brin et al. (1997) on the other hand used one single minimum support, which satisfied the downward closure property, but brought up the subject of how the items should be ordered in the data structure used for counting. According to Brin, having the items that occur in many itemsets to be last in the sort order of the items, and the items that occur in few itemsets to be last accomplishes more efficient counting. The specific researchers tried to apply an item reordering technique, whose results were rather disappointing, since the reordering played a negligible role in the overall performance, whereas sometimes it had the complete opposite effect, ordering the items completely reversal. The problem is especially apparent in datasets that are highly skewed, and localized changes or characteristics in data can have completely unpredictable results. The ordering that we propose in our algorithm first of all satisfies the sorted closure property suggested by Liu, Hsu and Ma (1999). Furthermore since according to this ordering the most frequent items are stored last in the order and the least frequent are stored first, we accomplish more effective counting, avoid the overhead incurred due to the re-sorting proposed in the work of Brin et al. (1997), and finally as shown in the final section, for highly skewed datasets our algorithm outer performs DIC.

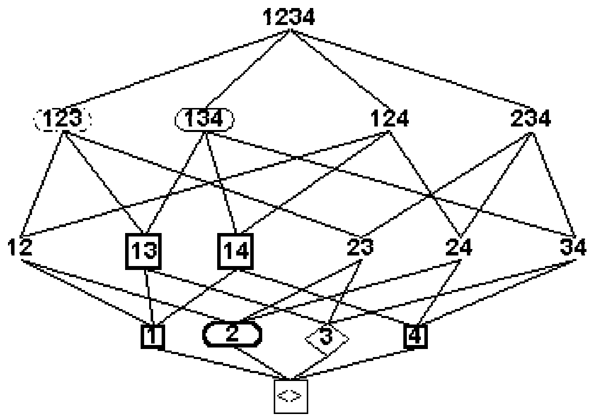


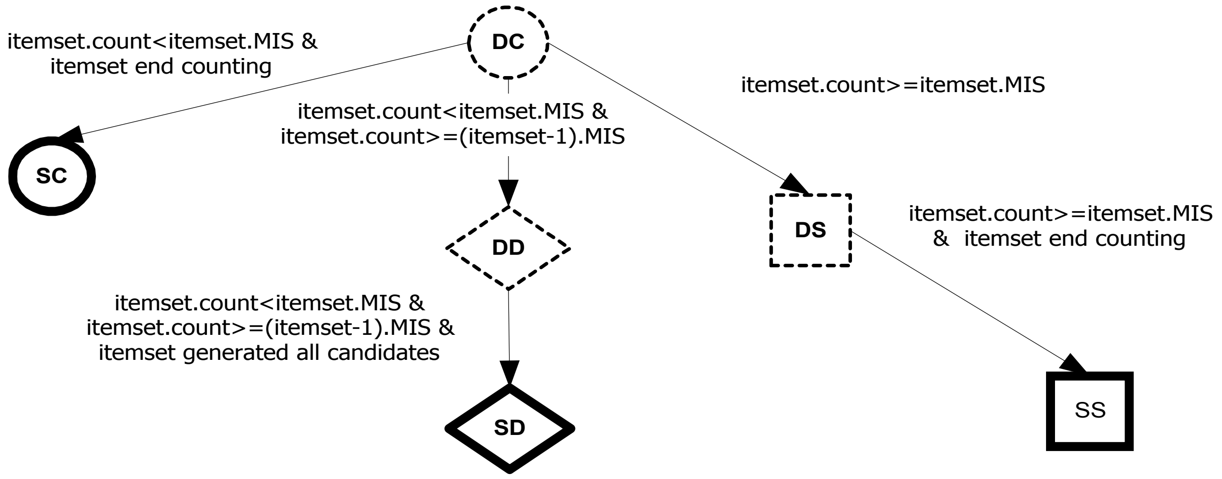
Figure 2: An itemsets lattice

The Data Structure

The data structure used by our algorithm is exactly like the hash tree used in DIC with a small but very significant difference. In every node of the tree we store also the corresponding MIS value of the specific node. This means that since we do not have a common support threshold for every itemset in the tree, each itemset must have its own support value. The MIS values for the 1-itemsets are assigned manually by the user, and from that point after the program itself assigns the MIS values to every node automatically. This step is given bellow:

```
1 for every itemset c=<c.item_1, c.item_2,...., c.item_k> that we begin counting
2 select c.item_1
3         c.MIS= c.item_1.MIS
```

Basically in every itemset that becomes a candidate (i.e. is stated DC) and starts to be counted we take the first item, which also has the lowest MIS value from all the others, and assign to the itemset this MIS value. In Figure 3 we present the symbols for the various itemset states, the transitions among the different states and finally when should these transitions occur. Managing transitions among all these states (from active to counted and from small to large) and detecting when these transitions should occur becomes a rather complicated task.



itemset.count<itemset.MIS & itemset end counting
DC
itemset.count>=itemset.MIS
itemset.count<itemset.MIS & itemset.count>=(itemset-1).MIS
SC
DD
DS
itemset.count>=itemset.MIS & itemset end counting
itemset.count<itemset.MIS & itemset.count>=(itemset-1).MIS & itemset generated all candidates
SD
SS

Figure 3: The transitions among the different states

Performance Comparison

In this section we compare our proposed method with algorithm MSApriori as well as with algorithm DIC. The experiments were run on a Celeron 333MHz PC, with 128MB of main memory, under the Windows 98 operating system.

Experiments with synthetic data

In order to carry out experiments with synthetic data we used the data generator[1], which is very well documented in (Agrawal and Srikant 1994). As stated by Liu, Hsu and Ma (1999) in the comparison of MSApriori to algorithm Apriori, as far as the synthetic data is concerned the time required by algorithm MSApriori is roughly the same as that needed by algorithm Apriori since the database scan dominates the computation. So we compared our algorithm directly to algorithm Apriori. The number of candidate and large itemsets are exactly the same for both algorithms and hence are not shown here.
In order to assign MIS values to the items in the data set we used the same method used by Liu, Hsu and Ma (1999), and more specifically we used the following formulas:

$$MIS(i)=\begin{cases} M(i) & M(i)>LS \\ LS & otherwise \end{cases} \quad where \quad \left| \begin{array}{c} M(i)=\beta f(i) \\ \beta=1/\alpha \end{array} \right.$$

Where f(i) is the actual frequency of an item i, LS is the user specified lowest minimum item support allowed and β is a parameter controlling how the MIS values for items should be related to their frequencies.

For our experiments we generated a number of data sets to test our proposed algorithm. All of the datasets had similar behavior, and so we present only one. However at its present version our algorithm does not support the automatic assignment of MIS values, and so everything had to be done manually. So we tested our algorithm with sets consisting of just some tenths of distinct items. We expect our algorithm to perform even better with more items. The data set shown here is generated using an average of 10 items per transaction, with 50 items and 100.000 transactions. The step length of our algorithm was fixed at 20% of the database (i.e. 20.000 transactions).

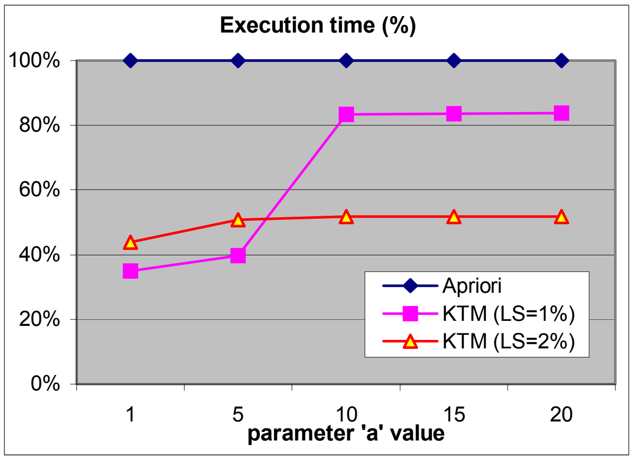


Figure 4: Comparison of execution times in percentage

In Figure 4 we show the comparison between the execution times of the two algorithms as a percentage. As we can see our algorithm manages to reduce the execution times considerably. For LS=1% our algorithm runs almost

[1] http://www.almaden.ibm.com/cs/quest/syndata.html

70% faster (for a=1), and for LS=2% almost 60% faster (for a=1).

Skewed Data

Algorithm DIC as well as our algorithm are both susceptible to data skew. Since there is no known generator for generating skewed synthetic datasets, that we are aware of, we decided to generate some custom data sets. We begun with a dataset consisting of 5 items and 100.000 transactions and gradually increased the number of items to 9.
The datasets that we created were so highly skewed that some itemsets appeared as seldom as only 100 times in a dataset of 100.000 transactions, all in consecutive transactions and most of the times at the end of the whole dataset. This means that sometimes both the algorithms had to wait until the end of the transaction file in order to begin counting some items, thus completely missing the advantage offered by both the algorithms which is starting to count higher order itemsets early enough in the dataset. This explains the slightly high times required by both algorithms. In these series of experiments we did not include also algorithm MSApriori as it would yield even higher execution times.
In these experiments, as can be seen also in Figure 5, algorithm KTM performed better than algorithm DIC in all cases, reducing the execution times from 2% in the dataset with 5 items to almost 10% in the dataset with 9 items and thus confirming what we have proposed previously.

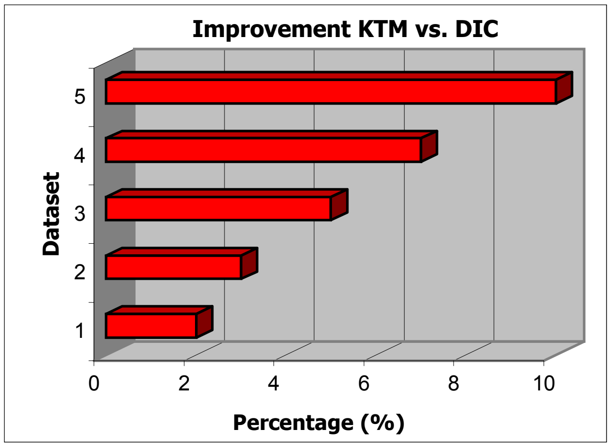


Figure 5: % Improvement of execution time

Conclusions and Future work

In the future we would like to run more extensive experiments in order to test the behavior of our algorithm. We would like to experiment more with features like the step length, the minimum confidence as well as with more distinct items. We would also like to test it with real data, preferably from a retail company.

References

Aggarwal, C.C. and Yu, P.S. 2001, 'A New Approach to Online Generation of Association Rules', IEEE Transactions on Knowledge and Data Engineering. Volume 13, No 4,pp. 527-540.

Agrawal, R., Imielinski, T. and Swani, A., 1993, 'Mining association rules between sets of items in very large databases', Proc. ACM SIGMOD Conf. Management of Data, pp.207-216.

Agrawal, R. and Srikant, S., 1994, 'Fast Algorithms for Mining Association Rules in Large Databases', Proc. of the 20th International Conf. on Very Large Data Bases..

Agrawal, R. and Srikant, S., 1995, 'Mining Sequential Patterns', Proc 11 Int'l Conf. Data Eng., pp.3-14.

Brin, S., Motwani, R, Ullman, J.D. and Tsur, S., 1997, 'Dynamic Itemset Counting and Implication Rules for Market Basket Data'. Proceedings of the ACM SIGMOD, pp. 265-276.

Chen, M.S., Han, J. and Yu, P.S., 'Data Mining: An overview from Database Perspective,' IBM Research Report RC 20556

Han, J. and Fu, Y., 1995, 'Discovery of multiple-level association rules from large databases.' VLDB-95.

Hidler, C., 1997, 'Online Association Rule Mining' Proc. ACM SIGMOD Int'l Conf. on Management of Data, ACM Press, New York, pp.171-182.

Kouris, I.N., Makris, C.H., and Tsakalidis, A.K., 2002, 'On-Line Generation of Association Rules using Inverted File Indexing and Compression', In Proc. IADIS Int'l WWW/Internet 2002 Conference, Lisbon, Portugal.

Lee, W., Stolfo, S. J., and Mok K. W., 1998, 'Mining audit data to built intrusion detection models.' KDD-98.

Liu, Bing, Hsu, Wynne and Ma, Yiming, 1999, 'Mining Association Rules with Multiple Minimum Supports'. KDD-99.

Mannila, H., 1998, 'Database methods for Data Mining' KDD-98 tutorial.

Park, J.S., Chen, M.S. and Yu, P.S., 1995, 'An Effective Hash Based Algorithm for Mining Association Rules', Proc. 1995 ACM SIGMOD Int'l Conf. Management of Data, pp. 175-186.

Srikant, R. and Agrawal, R., 1995, 'Mining Generalized Association Rules', Proc 21st Int'l Conf. Very Large Databases.

Toivonen, Hannu, 1996, 'Sampling Large Databases for Association Rules'. In Proc. of the 22nd Int'l Conf. on Very Large Data Bases, Mumbai (Bombay), India, pp.134-145.