ON-LINE GENERATION OF ASSOCIATION RULES USING INVERTED FILE INDEXING AND COMPRESSION

Ioannis N. Kouris  
Department of Computer Engineering and Informatics,  
University of Patras  
26500 Patras, Greece  
and  
Computer Technology Institute  
P.O. BOX 1192, 26110 Patras, Greece

Christos H. Makris  
Department of Computer Engineering and Informatics,  
University of Patras  
26500 Patras, Greece  
and  
Computer Technology Institute  
P.O. BOX 1192, 26110 Patras, Greece

Athanasios K. Tsakalidis  
Department of Computer Engineering and Informatics,  
University of Patras  
26500 Patras, Greece  
and  
Computer Technology Institute  
P.O. BOX 1192, 26110 Patras, Greece

ABSTRACT
We consider the problem of online mining of association rules in databases containing large numbers of transactions – and especially in market-basket data. We focus primarily on the use of a proper data structure that would allow us to handle large numbers of data for online mining of association rules, and also in presenting and giving answers to new types of online queries. We borrow techniques used in information retrieval, like inverted file indexing and inverted file compression. Thus, the online mining is accomplished by preprocessing the data once and creating an index by using an inverted file. Next, if this inverted file is too large to be held in main memory as is, we compress it. The inverted file is now ready to instantaneously give answers to various types of queries. We present the traditional kinds of queries to which our approach is capable of answering as well as we present some new kinds of queries that are supported. Finally we present how our approach could be used like a classic algorithm for finding large itemsets.

KEYWORDS
OLAP, association rules, data mining, knowledge discovery, inverted file indexing, index compression.

1. INTRODUCTION
Mining large databases for association rules has received great attention in the previous years. Much research has been made in the specific area, and many commercial products have been produced so far. Today the mining of such rules continues to be one of the most popular tasks in KDD (Knowledge Discovery in
Databases), both due to the fact that in many cases it involves direct financial interest (e.g., in market basket data), and also due to the fact that the data that is being produced, collected, stored, and needs to be analyzed, increases steadily.

The specific problem was first introduced by Agrawal et al. (1993), where the prototypical application was the analysis of supermarket sales or basket data and since then there have been made many attempts and approaches in order to improve various aspects of the specific problem. Various algorithms have been proposed in order to deal with this problem, other attempting to decrease the number of times the databases being read (Savasere et al. 1995), others to further decrease the number of candidate large itemsets (Park et al. 1995), and some others trying to reduce both (Brin et al. 1997).

Savasere et al. (1995) proposed an algorithm that would partition the database into disjoint partitions, with such sizes that could fit into main memory. Each partition would be searched and generate its own local large itemsets. The local large itemsets from each partition are then merged, and the global large itemsets are generated. After that the database is scanned once more and the support of the global large itemsets is found. The specific method requires only two passes over the entire database in order to find the large itemsets. Park et al. (1995) proposed an algorithm that worked liked a priori but would try to decrease the number of candidate itemsets, especially the 2-itemsets, by using a hash table structure to further prune candidate itemsets. Brin et al. (1997) suggested a new algorithm that would reduce the number of passes over the database, by beginning to count (k+n)-itemsets in parallel with k-itemsets. The specific method would require far fewer number of passes over the entire database. The idea of using some sampling techniques was implemented by Toivonen (1996), where he tried to reduce the number of passes over the database to two. The idea of using the TIDs (Transaction Identifiers), was first introduced by Agrawal and Srikant (1994). Following this work, the TIDs were also used by Savasere et al. (1995), as well as by Zaki et al. (1997). We are referring to the specific algorithms because the use of TID-lists is fundamental to our approach too. The mining of generalized association rules was presented by Agrawal et al. (1993), whereas Srikant and Agrawal (1996) established methods for mining quantitative association rules. For more details to the subject of data mining readers are referred to Chen et al. (1996), Hipp et al. (2000), Aggarwal (1998) and Yu.

Nevertheless almost all of the approaches have mostly an offline or batch behavior, rather than an online interactive one. The data miner would set some parameters, which in most cases are different for each algorithm (for association rule mining some are support and confidence), the system would process silently for a significant period and would then return some results. If we are not satisfied by the returned results the whole process would have to be repeated from the beginning and the data miner would have to “guess” some better values. It would be much better and easier if we gave the whole process a more interactive and online form, which would give the data miner more control and insight.

The idea of some form of online generation of association rules was first introduced by Aggarwal and Yu (2001). In the specific paper an algorithm resembling and adopting the principles of OLAP (On-line Analytical Processing) is proposed. The approach was to use a traditional algorithm (they used a hash based algorithm described by Park et al. 1995) to compute all large itemsets using some low support threshold s. The next step, which is the generation of the association rules, is done online by using a user-specified confidence and support threshold (which must be >= s). Another approach to the specific subject was made by Hidber (1997), who proposed a new algorithm (CARMA - Continuous Association Rule Mining Algorithm). According to this new algorithm, the database is scanned at most two times, and at any time during the first scan the user is able to control the process by dynamically changing the support threshold. During the second scan the small itemsets are pruned and the exact support for each large itemset is determined.

In the specific paper we adopt the main concept of OLAP, which is “preprocess the raw data once and query many times”, combine it with techniques from information retrieval, and propose a novel approach to this specific problem of data mining.

2. PROBLEM DESCRIPTION

We give a formal description of the problem based mainly on the statements made by Agrawal et al. (1993) and Agrawal and Srikant (1994), as well as based on what had been repeated in many works that followed Savasere et al. (1995), Park et al. (1995), Brin et al. (1997), Aggarwal and Yu (2001). Let I={i_1, i_2, ..., i_m} be a
set of literals called items. Let \( D \) be a set of transactions, where each transaction \( T \) is a variable length set of items such that \( T \subseteq I \). Since we do not take into account the quantities of each item bought in a transaction, each item is a binary variable representing if an item has been bought. We associate with each transaction a unique identifier, called its TID. Consider \( X \) to be a set of items in \( I \). A transaction \( T \) is said to contain \( X \) if and only if \( X \subseteq T \). Each itemset has an associated measure of statistical significance called support, which is defined as the fraction of transactions in \( D \) containing the specific itemset.

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

The problem of mining association rules is to generate all rules that have support and confidence greater than some user specified minimum support and confidence thresholds, respectively. This problem can be further decomposed into two distinct sub problems:

- Find all itemsets that have support above a user specified minimum support. These itemsets are called large, with every other itemset called small and discarded.
- Generate all association rules for the database that exceed a user specified minconfidence, using these large itemsets as input.

Clearly the step that is in essence performance critical is the first one, and it is the one that dominates the biggest part of the time. After the first step has been completed, i.e. after all the large itemsets have been discovered, the corresponding association rules can be generated easily (for more details refer to Agrawal and Srikant 1994).

3. KINDS OF ONLINE QUERIES

Following the work done by Aggarwal and Yu (2001), we also considered the queries that can and should be answered by the system that we propose, but this was as far as the traditional approaches and the traditional queries are concerned. Nevertheless none of the approaches up to now could give answers to queries like the ones posed in search engines or information retrieval systems. For example, it was not possible to give answers to questions like: How many people bought product A and product B but not product C, or for example how many people bought product A and B or product C, and many other combinations of the three Boolean operators (and – or – not), without wasting too much time and resources. The system that we propose is capable of achieving this instantaneously, by using minimum resources thus giving absolute freedom and flexibility to the data miner or the customer.

Furthermore, a great improvement of our system to the specific problem is that now we are capable of performing online mining of association rules at any given support and confidence level no matter how low it is. One of our main achievements is that we managed to disengage the generation of the large itemsets from the support property until the actual time a rule is being generated. According to Aggarwal and Yu (2001) it was only possible to perform online mining of association rules at support levels above the primary threshold. If we would like to mine bellow the specific threshold then we would have to process the entire dataset from the beginning, costing us both in time and other resources. In our case we do not face such a restriction since we store every single itemset, not taking into account the support or confidence levels until we actually generate some rules.

Another form of query our system is capable of answering is when we find all \((k+1)\)-itemsets containing one specific \( k \)-itemset, assisting that way a customer along his/her buys. According to this scenario, suppose we have an electronic shop selling products through some web site on the Internet. A customer browses through the products and at some point he puts product B in his basket. We now wish to find all the combinations of B with the other products and present him the most frequent ones. If there are \( N \) distinct itemsets in our dataset, there may be at max \( N-1 \) combinations of B with all the other itemsets, which we present to him. Suppose the same customer puts now product F in his basket. We now find all the combinations of itemset BF with all the other itemsets, which are at max \( N-2 \). As the customer continues to put products in his basket we continue to find all the new combinations. In Figure 1 we show all the
combinations that have to be performed in each step for the sequence of buys B – F – D – C, and for the set of products (A,B,C,D,E,F).

Figure 1. The combinations for the sequence of buys B – F – D – C for a customer.

Finally we are able to generate on the fly rules for specific itemsets, at any given level of support and confidence without checking the entire database or all their corresponding subsets.

4. INDEXING

In order to address the issue of how the data inside a database should be organized so that the various queries could be both efficiently and effectively resolved, especially in an online approach, we adopted techniques used in the classic information retrieval theory and more specifically we used inverted file indexing and compression techniques. For more details readers are referred to Witten et al. (1999).

5. COUNTING LARGE ITEMSETS

Apart from performing any online mining of association rules and answering to any kinds of queries, our algorithm is capable of working just like the traditional algorithms that are used for finding large itemsets Savasere et al. (1995) Agrawal and Srikant (1994) Aggarwal and Yu (1998). As mentioned earlier, the problem of mining association rules can be further decomposed into two distinct sub problems. First the problem of finding all itemsets above a user specified threshold and secondly the problem of generating all association rules using these large itemsets. We will deal with the first subproblem in the specific section.

Our algorithm, like all algorithms for finding large itemsets, relies on the property that an itemset can only be large if and only if all of its subsets are large, and so it starts by searching small itemsets. This observation results in the generation of a much smaller number of candidate itemsets by pruning itemsets that are bound not to be frequent in later iterations. Our algorithm proceeds level-wise too. First it counts all the 1-itemsets and returns those that are above the support threshold. Then it generates all the combinations of those large 1-itemsets, to create the candidate set of 2-itemsets. Those candidate 2-itemsets are counted and the large ones are found. The same happens with the 3-itemsets, i.e. the large 2-itemsets are combined to form the candidate 3-itemsets, and then the candidate 3-itemsets are counted and the large ones are identified. This continues until the largest itemset is found or until there are no more candidate itemsets. The factors that greatly influence the performance of this procedure are the number of passes made over all the data and the efficiency of those passes. Our approach tries not to reduce the number of passes made all over the data but rather to improve the efficiency of those passes. Let us see how.

First of all the database is scanned once and an index, using inverted files, is created. Then this index is compressed and stored in main memory for fast retrieval. This step is equivalent to the first step of the traditional algorithms, where the database is scanned once in order to find the candidate 1-itemsets and to count their appearances. The candidate 1-itemsets are already identified by our approach too, since the index stores every distinct itemset along with information about the number of transactions it appeared in and also which are those transactions. An example of an inverted list for an item is shown bellow.

\[ A = \langle 6, 2, 4, 5, 3, 7, 1 \rangle \]
From this candidate set we are now able to find which are the large 1-itemsets, by choosing those for which the number of appearances is above the user specified minimum. Combining these itemsets we form the candidate two itemsets. In order now to find out which are the large itemsets we query the index for each candidate 2-itemset using the Boolean operator AND. So if for example itemset AD appears on the candidate set of 2-itemsets we query the index by issuing the query A AND D. The two lists are intersected and the number of common appearances is returned. We continue querying the index for all itemsets, step by step. Since all the searching is done with the index stored in main memory it is obvious that the whole procedure is performed very fast.

**Example:** Suppose we have the sample database, which is shown in Figure 2. What we do is we replace the first scan of the database with a different procedure, and more specifically we create an inverted file index for the database, which we then compress and store in main memory. Additionally each consecutive scan of the database is replaced by the scan of the inverted file index, which is stored in main memory. After the first scan the database does not have to be scanned again. Suppose now that the user sets his support threshold to 2, meaning that in order an item to be considered large it must appear in at least two transactions. We can instantly generate the set of large itemsets \( L_1 \) by choosing from the compressed index those itemsets for which the number of appearances is at least two. In order to generate the set of candidate 2-itemsets we combine the large 1-itemsets in \( L_1 \) and we generate \( C_2 \). The support of each 2-itemset in \( C_2 \) is found by querying the index repeatedly. For example for itemset AB we query the index like this: A AND B. The two lists of A and B respectively are intersected and the support (the common items of the two lists) is returned. This is done until every itemset in \( C_2 \) is queried and their support is returned, thus creating \( L_2 \) - the set containing all large 2-itemsets. Combining the large 2-itemsets we create \( C_3 \) – the candidate set of 3-itemsets and so on.

As noted above one of the major improvements of our approach is that the creation of the index is completely disengaged from any statistical measure. The index is created irrespectively from what the support or any other measure used in other methods is, and does not change if we alter the support threshold. So if we are not satisfied with the output, for example too many or too few large itemsets are generated, we are able at any step and at any time to restart the whole process. The cost is significantly smaller compared to all other methods since everything is in main memory and the database does not have to be read again. What we have to do is re-query the index step by step using the new threshold. Using a traditional algorithm, like e.g. apriori (Agrawal and Srikant 1994), we would have to scan the whole database multiple times again.
6. EXPERIMENTAL RESULTS

In order to assess the viability and usefulness of our approach we conducted a series of experiments. We experimented primarily with artificial datasets (introduced by Agrawal and Srikant 1994), due to the reluctance and legal considerations of some retail companies to provide us with real data. For our experiments we used the synthetic data generator, which is very well documented by Agrawal and Srikant (1994), and is available from IBM\(^1\). The performance of our approach with various datasets is discussed firstly and then some scale-up experiments are presented.

One of our main concerns regarding our approach was the space and time needed in order to index each dataset. Would the final inverted file be too big in order to be stored in main memory, or would the time required to build it make it impractical? So the first experiments that we conducted tested the space and time requirements of our approach as far as the indexing phase was considered. We used a set of artificial datasets, in order to see what percentage of the input file was the final compressed index as well as what time was it needed in order to create the index for each dataset. The parameters used for the various datasets, namely parameters T, I and D stand for: T – Average size of transactions, I – Average size of maximal potentially large itemsets and D - Number of Transactions. The results are shown in Figure 3.

![Figure 3. Time needed to build and compress the index for various datasets](image)

In order to compress the index we used local Bernoulli model (Witten et al. 1999) with the Golomb code (Golomb 1966) to represent the gaps between the consecutive transactions (in our case noted as TIDs). Finally each inverted list is prefixed by an integer \( f \) to allow the calculation of the Golomb parameter \( b \). These integers \( (f) \) are represented using \( \alpha \)-coding.

Generally since our goal was not the level of compression of the index that we would accomplish but rather how fast it would be indexed and compressed we preferred a less complex method. Certainly there is a tradeoff between the complexity of the method and the final size of the inverted file but since we accomplish very small sizes of our inverted files we found little incentive in investing in more complex compression methods. The results for the various datasets are shown in Table 1.

\(^{1}\) http://www.almaden.ibm.com/cs/quest.syndata.html
Table 1. Size and percentage of compression of the inverted files for the various datasets

<table>
<thead>
<tr>
<th>DATASET</th>
<th>Initial size in (MB)</th>
<th>Inverted File size (MB)</th>
<th>% of Compression</th>
</tr>
</thead>
<tbody>
<tr>
<td>T5.I2.D100K</td>
<td>2.4</td>
<td>0.46</td>
<td>18.788</td>
</tr>
<tr>
<td>T5.I4.D100K</td>
<td>4.4</td>
<td>1.01</td>
<td>41.021</td>
</tr>
<tr>
<td>T5.I6.D100K</td>
<td>6.4</td>
<td>1.79</td>
<td>55.065</td>
</tr>
<tr>
<td>T10.I2.D100K</td>
<td>8.4</td>
<td>2.06</td>
<td>71.636</td>
</tr>
<tr>
<td>T10.I4.D100K</td>
<td>16.2</td>
<td>4.26</td>
<td>71.636</td>
</tr>
<tr>
<td>T10.I6.D100K</td>
<td>24.3</td>
<td>6.58</td>
<td>71.636</td>
</tr>
<tr>
<td>T20.I2.D100K</td>
<td>16.2</td>
<td>2.84</td>
<td>51.467</td>
</tr>
<tr>
<td>T20.I4.D100K</td>
<td>32.3</td>
<td>5.68</td>
<td>51.467</td>
</tr>
<tr>
<td>T20.I6.D100K</td>
<td>48.4</td>
<td>8.53</td>
<td>51.467</td>
</tr>
</tbody>
</table>

In order now to test the scale-up behavior of our approach we made a series of experiments, again with the synthetic data generator. We began first by varying the number of transactions from 100,000 to 2,000,000, for three different pair of values for $T$ and $|I|$. Specifically we used T5.I2, T10.I4, and T20.I6 and kept all other parameters the same as before. The results are shown in Figure 4 and Table 2.

Table 2. Size and percentage of compression of the inverted files for transactions scale-up

<table>
<thead>
<tr>
<th>DATASET</th>
<th>Transactions</th>
<th>Initial size in (MB)</th>
<th>Actual Size of Inverted File in MB</th>
<th>% of Compression</th>
</tr>
</thead>
<tbody>
<tr>
<td>T5.I2</td>
<td>250,000</td>
<td>8.1</td>
<td>1.42</td>
<td>18.78</td>
</tr>
<tr>
<td></td>
<td>500,000</td>
<td>16.2</td>
<td>2.84</td>
<td>18.77</td>
</tr>
<tr>
<td></td>
<td>750,000</td>
<td>24.3</td>
<td>4.26</td>
<td>18.78</td>
</tr>
<tr>
<td></td>
<td>1,000,000</td>
<td>32.3</td>
<td>5.68</td>
<td>18.78</td>
</tr>
<tr>
<td>T10.I4</td>
<td>250,000</td>
<td>15.2</td>
<td>2.58</td>
<td>17.364</td>
</tr>
<tr>
<td></td>
<td>500,000</td>
<td>30.4</td>
<td>5.16</td>
<td>17.367</td>
</tr>
<tr>
<td></td>
<td>750,000</td>
<td>45.6</td>
<td>7.73</td>
<td>17.367</td>
</tr>
<tr>
<td></td>
<td>1,000,000</td>
<td>60.9</td>
<td>10.31</td>
<td>17.366</td>
</tr>
<tr>
<td>T20.I6</td>
<td>250,000</td>
<td>29.5</td>
<td>4.56</td>
<td>15.489</td>
</tr>
<tr>
<td></td>
<td>500,000</td>
<td>59</td>
<td>9.13</td>
<td>15.488</td>
</tr>
<tr>
<td></td>
<td>750,000</td>
<td>88.5</td>
<td>13.69</td>
<td>15.489</td>
</tr>
<tr>
<td></td>
<td>1,000,000</td>
<td>118</td>
<td>18.28</td>
<td>15.489</td>
</tr>
</tbody>
</table>

7. CONCLUSION

Traditional market basket mining algorithms exhaustively explore the space of all possible association rules using a variety of properties like confidence, interest, and conviction (Brin et al. 1997). The problem with all these algorithms is that they require that we go through all the data in our database, some of them requiring as many passes, as is the higher order itemset. The approach that we propose is essentially different from all
current algorithms in that it requires only one pass over the data and does not produce any rules in advance but rather on the fly as each new customer makes his own transactions. The way it accomplishes that is by creating an inverted index by reading the dataset once, and then using this index each time a customer buys a product in order to propose items. The cost to create the index and to compress it can be justified by two facts. First according to Park et. al. (1995), it is essential to collect a sufficient amount of sales data (say, over last 30 days) before we can draw meaningful conclusions from them, and so the indexing procedure will not have to be done so frequently. Secondly the cost to preprocess the data can be amortized later by the increased flexibility and many queries that we are able to perform.

Another possible advantage of the proposed algorithm, is that it is less influenced by how correlated the data is. Most traditional algorithms work well with few distinct itemsets, which are not also highly correlated. Their performance though quickly degrades with datasets, that either have many distinct itemsets, or datasets in which there is a high degree of correlation among the data (like for example census data).

According to the approach made by Aggarwal and Yu (2001) it is not possible to perform online mining of association rules at support levels less than the primary threshold. In order to accomplish that, the data have to be reprocessed from the beginning with the new thresholds. In our approach there does not exist such a problem, since once the data is processed once without taking into account any measure or threshold, and then we are free to choose any threshold we like.

As said the collected data contains some hidden knowledge. This knowledge is the rules that can be generated after finding all the large itemsets. The problem is that inside the data there can be found hundreds, thousands or sometimes—even millions of useful or not rules. If one takes into account the redundancy of the rules generated from large itemsets (Aggarwal and Yu 2001), then the situation gets even worse. Why is it then necessary to find all those rules and not only those one’s that are more useful. Like itemsets, rules have some frequency of appearances too. For example if most people buy product B and C, then this rule has high confidence, and is also more probable that the new customers would also follow this rule and buy those items together. Instead of going through all the data and finding all the rules no matter how useful and important they are we are simply allowing the customers and the database to unveil it’s secrets with their transactions.

Furthermore the traditional algorithms require that we go through the entire dataset again and again each time we want to check it for updates. Our algorithm is less susceptible to updates in that it does not require all of the dataset to be searched again. All we have to do is update the index file from time to time. Of course at some point the whole index would have to be recompressed from the beginning, possibly using a new compression method.

Another problem arises with most of the traditional algorithms with the pruning technique. According to this technique, which was first introduced by Agrawal and Srikanth (1994), if an itemset is found to be non-frequent then all the higher order itemsets that include this itemset are bound to be non frequent too. So we simply forget everything that has to do with the specific itemset, since if some (k-)subset of an itemset \( l \in C_k \) does not belong to \( I_{k-1} \), then that itemset should be pruned from further consideration. But what happens if a customer begins a transaction or puts very early a non-frequent itemset in his basket? Then this customer with this specific transaction is doomed not to get any suggestions at all. Whatever he or she will buy from the moment on he chose a non-frequent itemset is going to be non-frequent too. But we would most certainly like to propose to a customer a non-so frequent item, even if with the traditional algorithms it falls below the specified minimum, than not to propose something at all. Our proposed algorithm overcomes this problem in that it does not have any measure for pruning some itemsets that are not so frequent. All it does is gets input from the customers and creates its own suggestions on the fly, ignoring any pruning technique. Of course the pruning technique could be incorporated very easily in our algorithm too, the same way it is used in the traditional algorithms.

Someone though would just say that this could be done with the traditional algorithms, by eliminating the use of the pruning technique at every stage of the process, but this is completely impractical. Say for example that we have a database with just 100 distinct items. Then if we did not use any pruning to eliminate non frequent higher order itemsets, according to \( \frac{\sum r}{r} = \frac{\sum r}{(\sum r)!r!} \) we would have to count as many as 4,950 2-itemsets, 161,700 3-itemsets, 3,921,225 4-itemsets and so on. So, we understand that the number of possible itemsets that have to be counted grows exponentially large, making any thoughts to eliminate the pruning technique prohibiting.
REFERENCES


