An Algorithm to Mine General Association Rules from Tabular Data

Siyamand Ayubi[1], Maybin Muyeba[2], John Keane[3]
[1] Faculty of engineering, University of Isfahan, Iran
[2] Liverpool Hope University, School of Computing, Liverpool, UK
[3] University of Manchester, School of Informatics, Manchester, UK
{s.ayubi@gmail.com, muyebam@hope.ac.uk, john.keane@manchester.ac.uk }

Abstract. Mining association rules is a major technique within data mining and has many applications. Most methods for mining association rules from tabular data mine simple rules which only represent equality in their items. Limiting the operator only to "=" results in many interesting frequent patterns that may exist not being identified. It is obvious that where there is an order between objects, greater than or less than a value is as important as equality. This motivates extension, from simple equality, to a more general set of operators. We address the problem of mining general association rules in tabular data where rules can have all operators $\{\leq, >, \neq, =\}$ in their antecedent part. The proposed algorithm, Mining General Rules (MGR), is applicable to datasets with discrete-ordered attributes and on quantitative discretized attributes. The proposed algorithm stores candidate general itemsets in a tree structure in such a way that supports of complex itemsets can be recursively computed from supports of simpler itemsets. The algorithm is shown to have benefits in terms of time complexity, memory management and has great potential for parallelization.

Keywords: data mining, general association rules, tabular data, equality operators.

1 Introduction

Association rule (AR) mining [1] has been traditionally applied to datasets of sales transactions (referred to as market basket data). A transaction 'T' is a set of items and contains an itemset 'I' if I $\subseteq$ T. If 'I' has k members, then 'I' is called a k_itemset. An AR is an implication $X \rightarrow Y$ where X and Y are itemsets with no items in common i.e. $X \cap Y = \varnothing$. The intuitive meaning of such a rule is that the transactions (or tuples) that contain X also contain Y. The rule $X \rightarrow Y$ holds with confidence c if c% of transactions that contain X also contain Y. The rule $X \rightarrow Y$ has a support s if s% of the transactions in the database contain X $\cup$ Y. Given a database, the problem of mining ARs is to generate all rules that have support and confidence greater than

the user-specified minimum thresholds, min-Support and min-Confidence. There are many algorithms to mine association rules from transactional data [6, 7]. The AR technique has also been applied to tabular datasets [2, 3, 5, 11]. The notation for an item is redefined in tabular datasets. Henceforth an item is a triple (a, Θ, v) where "a" is an attribute, "v" is the value of "a" and Θ is the operator between "a" and "v". An example of an AR in tabular data is as follows:

$(A_1 = 2)$ and $(A_2 = 3)$ and $(A_4 = 5) \Rightarrow A_6 = 1$ support = 10%, Confidence = 60%

where $A_{1,...,}A_6$ are attributes with equality operator "=" and referred to as *simple rules* [2, 9]. In some cases, simple rules are unable to show all hidden patterns of data. In situations where there are orders between values of attributes, greater than or less than a value is as important as equality. Simple rules have difficulties in extracting such patterns, their drawback being in dealing with quantitative attributes. Popularly, quantitative attributes can be discretized by partitioning domain values to base intervals, where the difficulty is selecting the number of base intervals. Too many intervals for a quantitative attribute means low support for single intervals. Hence some rules involving these attributes may not be found, on the other hand, partitioning values into too few intervals causes information to be lost. Some rules have maximum confidence only when some of the items in their antecedent part have small intervals. These problems can be solved to some extent by using $\{\leq, >, \neq, =\}$ operators in the items. The MinSup problem does not matter for these operators as the number of intervals has no effect on operators $\{\leq, >, \neq\}$. If the operators of items in a rule belong to $\{\leq, >, \neq, =\}$, then the rule is called a *general rule* [2, 9]. An example of such a rule is as follows:

$(A_1 > 2)$ *and* $(A_2 \neq 3)$ *and* $(A_4 = 5) \rightarrow (A_6 = 1)$ Support = 10%, Confidence = 60%

In this paper, we propose an algorithm, MGR (Mining General Rules), to mine general rules. The number of general itemsets is exponentially higher than the number of simple itemsets and it makes mining them too difficult. Thus, MGR mines general itemsets from simple itemsets by a recursive computation of simpler itemsets, all stored in a tree data structure. This feature systematically enables benefits in terms of time complexity, memory management and great potential for parallelization.

The paper is organized as follows: background and related work is given in section 2; section 3 presents needed terminology and notation; section 4 shows calculation of supports of general itemsets from simple ones; section 5 presents the MGR algorithm; experimental results are given in Section 6; and Section 7 considers further work and presents conclusions.

2 Background and Related Work

In this paper we are interested in a type of generalization introduced in [2]. This type of generalization extends the traditional equality operator used in normal

association rules to an operator set $\{\leq, >, \neq, =\}$. Its main characteristic is the high number of itemsets generated in regard to normal association rules. Hsu et al. [9] proposed an algorithm for mining such general rules from transactional data by extending algorithms for mining simple itemsets. This approach has poor performance on tabular data because of the high number of general itemsets.

The main problem of mining association rules when applied to tabular datasets is dealing with quantitative attributes, which are usually partitioned into intervals and treated as discrete values. However, specifying the number of intervals is difficult in this approach. In [11], consideration of the combination of base intervals as new intervals and extracting the itemsets for all the intervals was proposed. The approach has an acceptable result but in most cases, its time complexity is high.

There are approaches that do not require discretization of quantitative attributes. Some extract types of rules that are different to formal association rules [10, 13]. The others do not try to mine all general itemsets but rather focus on finding the best intervals for a given rule [4]. The input to these algorithms is a rule that does not have any interval for its quantitative items. The outputs are the best intervals for quantitative attributes of the given rule. In situations where we are interested in optimizing one rule, these approaches are very useful. However, they cannot be applied to mine all general rules as they have to scan data for each rule.

There are some methods that define some criteria for interestingness of the general rules and mine just a subset of the rules that satisfy those criteria. The method in [14] defines the pc_optimal set as the set of the most interesting general association rules and tries to find an approximation of it. Despite the good results of the approach on some datasets, it is not guaranteed to find good approximations of the pc_optimal set. Further, it is not proved that the pc_optimal set contains the whole set of interesting rules.

3 Terminology and Notation

The following formal definitions are used to describe our approach and to prove related Lemmas.

Definition 1*(I): The set of all attributes of a table.*

We assume that attribute values are finite and discrete and there is an order between them. The finiteness assumption of attribute values is not restrictive because by partitioning infinite sets into subsets, we can easily convert infinite domains to finite domains. We use a_i for the ith attribute in I and $V_{i,j}$ for jth value of ith attribute. As there is an order between the values of each attribute, $V_{i,1} < V_{i,2}, \ldots, < V_{i,n}$.

Definition 2 *(Item): An item is a triple (a, Θ, v) where "a" is an attribute, "v" is a value of "a" and Θ is an operator between "a" and "v". An item is a simple item if its operator is "=". An item is a half general item, if the operator Θ is one of* $\{\leq, =\}$. *An item is a general item if the operator Θ is one of* $\{\leq, >, \neq, =\}$.

Note that the Θ cannot be $\{\geq, <\}$operators. As items have discrete values, we need not contain such operators e.g. ($i \geq j \Leftrightarrow i > j-1$) and ($i < j \Leftrightarrow i \leq j-1$).

Definition 3 *(Itemset): An itemset is the set of items (from definition 2). If every item in the itemset is a simple item, the itemset is a simple itemset. If every item in the itemset is a half general item, the itemset is a half general itemset. If every item in an itemset is a general item, the itemset is a general itemset.*

Definition 4 (t(X)): *t(X) is the set of IDs of records in a dataset which match the itemset X.*

4 Finding Supports of General Itemsets

Before we describe the MGR algorithm for mining general rules and itemsets, we first prove that we can obtain the supports of half general and general itemsets from simple and half general ones respectively. Lemma 2 explains the fact that the support of each half general itemset can be calculated from supports of simpler itemsets. Lemmas 3 and 4 explain the calculation of supports of general itemsets from half general ones.

Lemma 1: Let X be a half general itemset where its ith item has "$\leq$" as an operator and $V_{i,1}$ as a value. The support of X does not change if we convert the operator of the ith item to "=" and vice versa.

Proof: Because $V_{i,1}$ is the smallest value for the ith attribute, there are no records in the dataset with a smaller value than it, therefore all records in t(X) must have the value $V_{i,1}$ for the ith attribute. So changing operator "$\leq$" to "=" and vice versa does not change the support of the itemset.

Lemma 2: *Let X, Y and Z be half general itemsets that differ only in their ith item. They have the same attribute for the ith item but the operator of the ith item of X and Y is "$\leq$" and the operator of the ith item of Z is "=". The value of the ith item of X and Z is $V_{i,j}$ and the value of the ith item of Y is $V_{i,j-1}$, so the ith item in X and Z has one higher value than the ith item in Y. Then the supports of these itemsets have the following relationship:*

$$Sup(X) = Sup(Y) + Sup(Z)$$

Proof: Let a_i be the attribute of the ith item of X, Y and Z. As the ith item of X is $(a_i \leq V_{i,j})$, so for each $r \in t(X)$, we have $r(a_i) = V_{i,j}$ or $r(a_i) < V_{i,j}$ where $r(a_i)$ is the value of r for attribute a_i. Then t(X) can be partitioned into two subsets $t(X_1)$ and $t(X_2)$ according to the value of a_i such that the itemset X_1 contains $a_i=V_{i,j}$, and the itemset X_2 contains $a_i \leq V_{i,j-1}$ where the $V_{i,j-1}$ is the value before $V_{i,j}$. Therefore $t(X) = t(X_1) \cup t(X_2)$. As the operator of the ith item of Z is "=", then the itemset Z is

equal to X_1. We can prove the same for itemsets Y and X_2. Therefore we have t(X) = t(Y) ∪ t(Z), $t(Y) \cap t(Z) = \phi$ and consequently we have sup(X) = sup(Y) + sup(Z).

Table 1. A simple dataset with four Attributes {A_1, A_2, A_3, A_4}

Record Id	A1	A2	A3	A4
1	1	1	1	1
2	1	1	2	1
3	1	1	2	0
4	2	2	1	0
5	2	3	2	1
6	2	2	3	1
7	3	2	3	1
8	3	2	4	0
9	2	3	2	0
10	4	3	4	1
11	4	4	5	1

Example 1: *Suppose* we have a dataset as in Table 1, and the itemsets X, Y and Z have the following definitions:

$$X = (A_1 \le 2) and (A_2 \le 3) and (A_3 \le 2)$$
$$Y = (A_1 \le 2) and (A_2 \le 2) and (A_3 \le 2)$$
$$Z = (A_1 \le 2) and (A_2 = 3) and (A_3 \le 2)$$

Lemma 2 proves that *Sup(X) = Sup(Y) + Sup(Z).* As can be seen, the three itemsets differ in their second items. According to Lemma 2, we can partition X into the following itemsets:

$$X_1 = (A_1 \le 2) and (A_2 \le 2) and (A_3 \le 2)$$
$$X_2 = (A_1 \le 2) and (A_2 = 3) and (A_3 \le 2)$$

Hence $t(X_1) = \{1,2,3,4\}$, Sup(X_1) =4, $t(X_2) = \{5,9\}$,and Sup(X_2)=2.

As $t(X) = t(X_1) \cup t(X_2)$ then t(X) = {1,2,3,4,5,9} and Sup(X)=Sup(X_1) + Sup(X_2). As a result, Y=X_1 and Z=X_2, therefore Sup(X) =Sup(Y) + Sup(Z).

By using Lemma 1 and Lemma 2, the support of each half general itemset can be calculated from supports of simpler itemsets. In other words, if X is a half general itemset, then by applying Lemma 1 and Lemma 2 recursively, t(X) can be partitioned into t(X_1) ,.., t(X_n) where each X_i is a simple itemset.

Lemma 3: If X is a general itemset that does not have any item with an attribute a_i, then

$$Sup(X \cup (a_i > V_{i,j})) = Sup(X) - Sup(X \cup (a_i \le V_{i,j}))$$

Proof: Records of t(X) can be partitioned into two subsets such that t(X) = t(X_1) ∪ t(X_2), where t(X_1) is the set of all records that have a value greater than $V_{i,j}$. (t(X_1) = t(X ∪ (ai > $V_{i,j}$))) and t(X_2) is the set of all records that has a value equal to or smaller than $V_{i,j}$, (t(X_2) = t(X ∪ (ai ≤ $V_{i,j}$))). Then we have

$$t(X) = t(X \cup (a_i \le V_{i,j})) \cup t(X \cup (a_i > V_{i,j})) \text{ or}$$
$$Sup(X) = sup(X \cup (a_i > V_{i,j})) + Sup(X \cup (a_i \le V_{i,j})) \text{ or}$$

$$Sup(X \cup (a_i > V_{i,j})) = Sup(X) - Sup(X \cup (a_i \leq V_{i,j}))$$

Lemma 4*: If X is a general itemset that does not have any item with the attribute a_i, then*

$$Sup(X \cup (a_i <> V_{i,j})) = Sup(X) - Sup(X \cup (a_i = V_{i,j}))$$

Proof: The proof is similar to that of Lemma 3.

Example 2 *If we extract itemsets X, Y and Z from Table 1, with the following definitions:*

$$X = (A_1 \leq 2) and (A_2 \leq 3) and (A_3 > 2)$$
$$Y = (A_1 \leq 2) and (A_2 \leq 3) and (A_3 \leq 2)$$
$$Z = (A_1 \leq 2) and (A_2 \leq 3)$$

then Lemma 3 proves that *Sup(X) = Sup(Z) - Sup(Y).* According to Table 1, we have:

t(X)={6}, t(Y)={12,3,4,5,9}, t(Z)={1,2,3,4,5,6,9}
t(X)=t(Z)-t(Y), so $Sup(X) = Sup(Z) - Sup(Y)$.

5 The MGR Algorithm

Although the number of general itemsets is exponentially higher than the number of simple itemsets, by applying the Lemmas of the previous section on itemsets in a systematic way, this enables the MGR algorithm to divide the problem into smaller ones and solve them more quickly. The main steps of the algorithm are as follows:

i. Mining simple itemsets using one of the existing algorithms.
ii. Mining half general itemsets from simple itemsets.
iii. Mining general itemsets from half general itemsets.
iv. Mining general rules from general itemsets.

The first step is achieved by using one of the existing methods for mining simple itemsets. The last step is similar to other association rule algorithms and we do not focus on it here. The main steps of the algorithm are steps 2 and 3. The MGR algorithm does these steps by applying Lemmas 1, 2, 3 and 4 on itemsets in a tree data structure called an MGR tree.

The MGR tree brings two benefits in mining general itemsets. Firstly, it facilitates finding itemsets in such a way that the Lemmas of section 4 can be applied more easily. Secondly, it breaks the problem of mining general itemsets from simpler problems. Before describing the MGR algorithm, we first explain the structure of the MGR tree.

The root of the MGR tree contains no data. The nodes of the first level of the tree are called *signatures*. All itemsets inside a signature have the same attributes. It is designed so in order to facilitate applying Lemma 2, as itemsets in Lemma 2 have the same attributes. Nodes of the second level of the tree are called Half General Itemsets (HGI) nodes. Itemsets inside an HGI node have the same values for corresponding items, so each HGI node has just one simple itemset. Nodes of the last level of the

tree are called GI (General Itemset) nodes. Each GI node has just one half general itemset. Itemsets inside each GI node can be created from its half general itemset by converting operators "=" and "≤" to operators "≠" and ">" respectively.

5.1 Mining Half General itemsets

At this step of the algorithm, the extracted simple itemsets are partitioned into signatures. By defining a lexographical order between attributes and using the order between values of each attribute, the simple itemsets in each signature can be sorted, which is very important in finding itemsets. In the second level of the tree, there are HGI nodes which contain half general itemsets. Each simple itemset corresponds to an HGI node at this level of the tree. Half general itemsets of an HGI node have the same values for corresponding items, the only difference being the item operators. Mining half general itemsets in each signature begins from the first HGI node and up to the last one. The crucial issue here is that HGI node itemsets must be created in order according to figure 2. This enables the MGR algorithm to have random access to itemsets of an HGI node. Figure 1 shows the process of mining half general itemsets in each signature.

5.1.1 Illustrative Example of Mining Half General Itemsets

In this section, we illustrate the process of calculating the supports of half general itemsets. Suppose that itemsets of figure 3 are simple itemsets that belong to the signature $\{A_1, A_2\}$ and we want to calculate the supports of the half general itemsets of the signature. Figure 4 shows the HGI node of the first simple itemset. This HGI node is the first one that must be taken into account. The first itemset of this node is a simple itemset and its support is known. The supports of the other itemsets of the node are calculated by applying Lemma 1. Now let's consider the next HGI node which is shown in figure 5. The support of the first itemset is given (simple itemset). Support of itemset K_2 is calculated from the supports of itemsets K_1 and I_2 by applying Lemma 1. Support of itemset K_3 is calculated by applying Lemma 1. The same process can be done for other HGI nodes. In fact, we use the itemsets of the previous HGI nodes to calculate supports of an HGI node. If one of the required itemset does not exist, the process will be repeated to calculate its support. We suppose that absent simple itemsets have zero supports.

MineHalfGeneralItemsets (Signature)
1)Sort Simple Itemsets of the Signature;
2) For each simple itemset
Create corresponding HGI node;
3) For each HGI node
Create the half general itemsets in order using the method of figure 2;
4) For each half general itemset
Calculate support using Lemmas 1, 2

Fig. 1. Procedure: Mining half general itemsets inside a signature

by considering the operators '=' ,'<=' as the lower rank operators and by assuming that items of itemsets are sorted based on the attributes, we can calculate the address of each general itemset inside the GI nodes.

$(A_1 = 3) and (A_2 \leq 4) and (A_3 <= 2)$ 000 Address = 0

$(A_1 = 3) and (A_2 \leq 4) and (A_3 > 2)$ 001 Address = 1

$(A_1 = 3) and (A_2 > 4) and (A_3 <= 2)$ 010 Address = 2

$(A_1 = 3) and (A_2 > 4) and (A_3 > 2)$ 011 Address = 3

$(A_1 \neq 3) and (A_2 \leq 4) and (A_3 <= 2)$ 100 Address = 4

$(A_1 \neq 3) and (A_2 \leq 4) and (A_3 > 2)$ 101 Address = 5

Fig. 2. Half general itemsets inside an HGI node

$(A_1 = 1) and (A_2 = 1)$ Support =10%

$(A_1 = 1) and (A_2 = 2)$ Support = 7%

$(A_1 = 1) and (A_2 = 3)$ Support =8%

$(A_1 = 2) and (A_2 = 1)$ Support =15%

$(A_1 = 2) and (A_2 = 2)$ Support = 31%

$(A_1 = 2) and (A_2 = 3)$ Support = 12%

$(A_1 = 3) and (A_2 = 1)$ Support = 7%

$(A_1 = 3) and (A_2 = 2)$ Support = 10%

Fig. 3. Simple itemsets of the Signature $\{A_1, A_2\}$

I_1 $(A_1 = 1) and (A_2 = 1)$ Support =10%

I_2 $(A_1 = 1) and (A_2 \leq 1)$ Support =10%

I_3 $(A_1 \leq 1) and (A_2 = 1)$ Support =10%

I_4 $(A_1 \leq 1) and (A_2 \leq 1)$ Support =10%

Fig. 4. Itemsets of the first HGI node

K_1 $(A_1 = 1) and (A_2 = 2)$ Support =7%

K_2 $(A_1 = 1) and (A_2 \leq 2)$ Support =17%

K_3 $(A_1 \leq 1) and (A_2 = 2)$ Support =7%

K_4 $(A_1 \leq 1) and (A_2 \leq 2)$ Support =17%

Fig. 5. Itemsets of the second HGI node

5.2 Mining General Itemsets

The process of mining general itemsets from half general itemsets is similar to the process of mining half general itemsets from simple ones. For each extracted half general itemset, a GI node (General Itemset node) will be created. These GI nodes will contain general itemsets that can be created from the half general itemsets by converting operators (=, <=) to operators (!=, >) respectively. Similar to the process of creating half general itemsets, we can generate general itemsets inside GI nodes in such a way that we can have random access to them. This process is done by considering operators (=, <=) as low rank and by assuming that items of itemsets are sorted based on the attribute ranking. Figure 7 illustrates the process of creating general itemsets inside a GI node. The corresponding half general itemset of the node

in the figure is *'(A_1=3) and (A_2≤ 4) and (A_3≤ 2)'*. It is the first itemset of the GI node. As this itemset has operators {=, ≤, ≤} (low rank operators), its address in the GI node will be 0. The next itemset will be generated from the above itemset by converting the operator of the last item from ≤ to >. The created itemset has operators {=, ≤, >} for corresponding items (high rank operator for the last item), so its address in the GI node will be 001 or 1. The next itemset will have operators {=, >, ≤}, so its address in the GI node will be 010 or 2 (high rank operator for the middle item). The fourth generated itemset is *'(A_1=3) and (A_2>4) and (A_3>2)'* which has operators {=, >, >} and is located at address (011) (high rank operators for the last two items). The other itemsets will be created in the same manner.

After generating itemsets inside GI nodes, the next step is to calculate the supports of itemsets. The main difference between calculating supports of half general itemsets and general itemsets is the fact that each GI node must have addresses of its parents. The parents of a k-itemset are the k-1 subsets. For example, *(A_1=3)* is a parent of the itemset *(A_1=3) and (A_2=2)*. Having addresses of the parents facilitates the application of Lemmas 3 and 4. Figure 6 shows an algorithm to extract general itemsets.

MineGeneralItemset (Signature)
1) for each half general Itemset:
- Create Corresponding GI node
- Find the address of its parents

2) For each HGI node :
- For each GI node:
  - Create general itemsets according to figure 4;
  - Calculate the support of each general itemset using Lemmas 3 and 4
  -

Fig. 6. The procedure of mining general itemsets of a signature

by considering the operators '=' ,'<=' as the lower rank operators and by assuming that items of itemsets are sorted based on the attributes, we can calculate the address of each general itemset inside the GI nodes.

$(A_1 = 3) and (A_2 \le 4) and (A_3 <= 2)$	000	Address = 0
$(A_1 = 3) and (A_2 \le 4) and (A_3 > 2)$	001	Address =1
$(A_1 = 3) and (A_2 > 4) and (A_3 <= 2)$	010	Address =2
$(A_1 = 3) and (A_2 > 4) and (A_3 > 2)$	011	Address=3
$(A_1 \ne 3) and (A_2 \le 4) and (A_3 <= 2)$	100	Address =4
$(A_1 \ne 3) and (A_2 \le 4) and (A_3 > 2)$	101	Address =5

Fig. 7. General itemsets inside a GI node

5.3 Time Complexity and Memory Management of the MGR Algorithm

Mining half general itemsets from simple itemsets is approximately of linear complexity with regard to the number of simple itemsets. For each simple k_itemset, there are 2^k half general itemsets. According to Lemma 2, in order to compute the support of each half general itemset we need to search two half general itemsets. The complexity of searching one of them is log(s) where s is the average number of itemsets in signatures. The other itemset is located in the same HGI node as we have random access to half general itemsets of an HGI node, so its time complexity is negligible. Hence the overall time complexity of mining half general itemsets is equal

to $2^k n(\log_2 s)$ where n is the number of simple itemsets and k is a constant with its maximum value equal to the number of attributes. Here $2^k n$ is the average number of half general itemsets.

The average number of half general itemsets in signatures can be calculated from the following relation:

s= (total number of half general itemsets)/ (number of signatures)

where (number of signatures) $= 2^{|I|} - 1$,(number of half general itemsets) $\leq n.2^{|I|}$
To generate general itemsets, we should first sort the signatures ($(2^{|I|} - 1)\log(2^{|I|} - 1)$). Then according to the section 5.2, we should find all the k-1_itemsets for each half general k_itemset ($2^k n.k.\log_2 s$), finally the algorithm must calculate the supports of all half general itemsets ($4^k n2^{k-1}(1+k)$).

So, the overall time complexity of the MGR algorithm is equal to

$\Theta(n\log_2 s) + \Theta(n\log_2 n) + \Theta(n)$ or by substituting s,

$\Theta(n\log_2 n) + \Theta(n\log_2 n) + \Theta(n)$ or in fact $\Theta(n.\log n)$.

The other advantage of using an MGR tree is partitioning the problem into smaller ones. As seen in previous sections, mining itemsets in each signature can be done independently to the other signatures. It means that in each phase of mining itemsets, only holding one signature and its ancestors in main memory is sufficient. From the memory management point of view, it means that the MGR algorithm can be applied to large datasets without the need to hold all itemsets in main memory. From a parallel processing point of view, it means that mining itemsets in signatures can be done by different processors without any deadlock.

6 Experimental Results

6.1 Requirements of the Algorithm

The first step of the MGR algorithm, which is about mining simple itemsets, has great effect on the output of the algorithm. If there is no restriction on the supports of simple itemsets, the MGR algorithm can extract all general itemsets. If we set a non-zero value for support-threshold of simple itemsets, some infrequent itemsets will not be presented to the MGR algorithm. Absence of these itemsets has two effects on the output of the algorithm. Firstly, the algorithm ignores constructing HGI nodes corresponding to those simple itemsets which leads to the loss of all general itemsets belonging to those HGI nodes. Secondly, it causes errors in calculating the supports of general itemsets because the absent itemsets have an effect on supports of general itemsets. In order to achieve high quality rules, the support-threshold of simple itemsets must be low. In the experimental results of the following sections, the support threshold of simple itemsets is set to zero.

6.2 Experimental Results

In order to present the efficiency of the MGR algorithm, we compare it with an extension of the Apriori algorithm which mines general itemsets over combinations of base intervals similar to [11]. The Apriori algorithm is implemented using the Trie data structure and has better performance than many known algorithms [4]. It is designed so that its output is similar to the output of the MGR algorithm. This helps effective comparison of the algorithms. Extending the FP-growth algorithm [7] to mine general itemsets has difficulties because each branch of the FP-tree will contain items with similar attributes. For example, a record that has value 1 for attribute "A_1" can cover items {(A_1=1), (A_1<=1), (A_1<=2), (A_1<=3), (A_1<=4), (A_1<>2), (A_1<>3),..} etc.

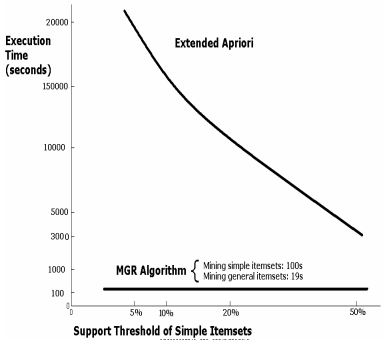

Fig. 8. MGR Vs Extended Apriori on synthetic dataset

Table 2. Properties of the synthetic dataset

Number of non class attributes	4
Domain values of non-class attributes	{0,1,2,3,4}
Number of classes	3
Number of records	100,000
Error ratio	0.1
Missing values ratio	0

All such items will exist in each branch of the FP-tree which causes trouble, as each itemset must have just one item with A_1 attribute. In order to avoid such difficulties, we do not use the FP-growth algorithm. We apply the algorithms on a synthetic dataset which is created using the DataGen tool [12]. Table 2 contains the details of the dataset. Figure 8 represents the execution time when applying both the extended Apriori and MGR algorithms to extract general itemsets from the synthetic dataset. The support threshold of simple itemsets using the MGR algorithm was set to zero.

As can be seen, the execution time of the MGR algorithm on the synthetic dataset is more than an order of magnitude lower than for extended Apriori. The total execution time of MGR is less than 120 seconds, while the execution time of extended Apriori with supports higher than 5% is about 21167 seconds. It can be inferred that the performance of the MGR algorithm is independent of the number of records or the size of the dataset (see figures 9 and 10). Figures 9 and 10 represent the execution time of the MGR algorithm with respect to the number of records. Figure 9 shows the total execution time of the MGR algorithm, which consists of mining simple itemsets and mining general itemsets. Figure 10 only shows the execution time

for mining general itemsets from simple ones. As can be seen, the total execution time of the algorithm is almost linear irrespective of the number of records. It is easily inferred from figures 9 and 10 that mining general itemsets from simple itemsets is approximately constant and in fact, it is mining simple itemsets that is linear with regards to the number of records.

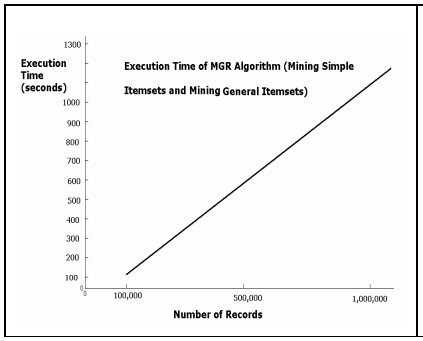

Fig. 9. Total MGR execution time

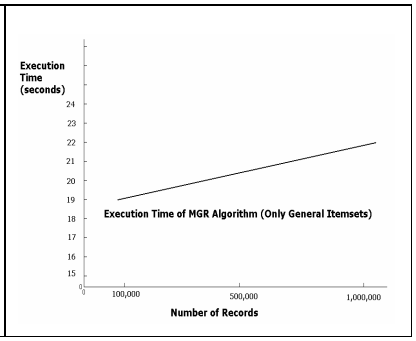

Fig. 10. Mining general items

7 Conclusions

In this paper, we proposed a time and space-efficient new algorithm for mining general association rules from tabular data. Decomposing the problem into several sub problems and employing the MGR tree makes the algorithm efficient in terms of time-complexity and memory requirements. The possibility of holding most of the MGR tree in secondary memory (hard disk) also makes the algorithm more space-efficient. In particular, it was shown that the algorithm stores candidate general itemsets in a tree structure in such a way that supports of complex itemsets can be recursively computed from supports of simpler itemsets.

As general rules can have equality and other comparison operators like $\{\leq, \geq, <, >, \neq, =\}$, we can discover more sophisticated patterns in data. As general rules have higher support and confidence than simple ones, they can represent more powerful patterns. In this paper, we have shown the power of general association rules to describe data, however we have not yet offered an approach to prune unnecessary general rules. More experiments will be done to compare the general and simple rules extracted from the Balance dataset [8]. In addition, further work will address ways for pruning insignificant rules as well as the potential for parallelizing the algorithm.

References

1. Agrawal, R., Imielinski, T., Swami, A.: Mining Association Rules between Sets of Items in Large Databases. In: SIGMOD Conference, pp. 207--216 (1993)

2. Richards, G., Rayward-Smith, V. J.: Discovery of Association Rules in Tabular Data. In: IEEE International Conference on Data Mining, (2001)
3. Berzal, F., Cubero, J-C., Marin, N., Serrano, J.: TBAR: an Efficient Method for Association Rule Mining in Relational Databases. Data and Knowledge Engineering. 37, pp. 47--64 (2001)
4. Bodon, F.: A Trie-based APRIORI Implementation for Mining Frequent Item Sequences. In: 1st International Workshop on Open Source Data Mining: Frequent Pattern Mining Implementations, pp. 56--65 (2005)
5. Rastogi. R., Shim, K.: Mining Optimized Support Rules for Numeric Attributes. In: 15th International Conference on Data Engineering, IEEE Computer Society Press Sydney, Australia, pp 126--135 (1999)
6. Agrawal, R., Srikant, R.: Fast Algorithms for Mining Association Rules. In: VLDB Conference, pp. 487--499 (1994)
7. Han, J., Pei, J., Yin, Y.: Mining Frequent Patterns without Candidate Generation. In: ACM SIGMOD Conference, pp. 1--12 (2000)
8. Merz, C. J., Murphy, P.: UCI Repository of Machine Learning Databases. (http://www.cs.uci.edu/mlearn/MLRepository.html), (1996)
9. Hsu, P., Chen, Y., Ling, C-C.: Algorithms for Mining Association Rules in Bag Databases. Information Sciences 166 (1-4), 31--47 (2004)
10. Aumann, Y., Lindell, Y.: A Statistical Theory for Quantitative Association Rules. Journal of Intelligent Information Systems, 20 (3), 255--283 (2003)
11. Sirkant, R., Agrawal, R.: Mining Quantitative Association Rules in Large Relational Tables, SIGMOD, 1--12 (1996).
12. Melli, G.: DataGen Tool, (http://www.datasetgenerator.com/), (2004)
13. Ke, J., Cheng, J., Ng, W.: Mining Quantitative Correlated Patterns using an Information Theoretic Approach. In: 12th ACM SIGKDD KDD Conference, Philadelphia, PA, USA, pp. 227--236 (2006)
14. Richards, G., Rayward-Smith, V. J..: The Discovery of Association Rules from Tabular Databases Comprising Nominal and Ordinal Attributes" in Intelligent Data Analysis, vol. 9, No. 3, pp. 289--307 (2005).