Graph-Clustering Association Rules Mining Algorithm

Conference Paper · June 2010

CITATIONS
0

READS
69

2 authors, including:

Amer Al-Badarneh
Jordan University of Science and Technology
59 PUBLICATIONS **184** CITATIONS

SEE PROFILE

Graph-Clustering Association Rules Mining Algorithm

Amer AL-Badarneh [†] Jamal Sakran

Department of Computer Science

Jordan University of Science and Technology

Irbid 22110, Jordan

{ amerb, sakrancs}@just.edu.jo

Abstract

Association rules mining algorithms have been extensively researched in the last decade. This paper proposes an efficient algorithm for association rules mining called GCAR. The proposed algorithm employs both graph and clustering techniques to discover association rules. The graph technique reduces the database scans, while the clustering technique eliminates some candidate itemsets that cannot be frequent. From a practical point of view, GCAR is very attractive because it reduces the number of data scans, eliminates some infrequent candidate itemsets, and hence has better performance. Several experiments on real data as well as synthetic data showed that GCAR outperforms the most well known association rules algorithm Apriori.

Keywords: association rules mining, data mining, Apriori, clustering.

[†] To whom correspondence should be addressed

1. Introduction

Association rules mining algorithms discover interesting relationships between data items that occur frequently together. Since their introduction in 1993 by Argawal et al. [1], the association rules mining problems have received a great attention. Within the past decade, hundreds of research papers have been published presenting new algorithms or improvements on existing algorithms to solve mining problems more efficiently.

Many applications took the benefit of association rules mining to improve the decision making process, such as market basket analysis, catalog design, cross marketing, and sales transaction. Market basket analysis is a typical example of association rules analysis that discovers buying behavior of customers. The discovered association rules can help decision makers develop marketing strategies.

Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of items. Let D be a set of transactions, where each transaction $T \subseteq I$ is a set of items. Associated with each transaction is a unique identifier, called *TID*. An association rule is an implication of the form $X \Rightarrow Y$, where $X \subset I$, $Y \subset I$, and $X \cap Y = \Phi$. 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 rules that have support and confidence greater than the user-specified minimum support (denoted as *min-sup*) and the user minimum confidence (denoted as *min-conf*) are called *interesting rules*.

A set of items is referred to as an *itemset*; *k-itemset* is an itemset that contains k items. Associated with each itemset is a counter that holds the support of that itemset. *Frequent itemset* is an itemset that has support greater than *min-sup*. *Candidate itemset* is an itemset that is expected to be frequent. In the rest of this

paper, we assume that the items are from ordered set and the transaction's items are sorted.

The problem of mining association rules consists of two steps: First, finding all frequent itemsets that have support above a predefined support value. The overall performance of mining association rules algorithms depends on this step. Second, generating the association rules using the frequent itemsets. This is a straightforward step. The efficiency of association rules mining algorithms depends mainly on two main factors: the number of passes over the data and the way candidate itemsets are generated through each pass. That is, the overall performance of the association rules algorithms can be improved efficiently by optimizing any of these factors.

This paper proposes a new association rules mining algorithm, called GCAR, that employs both graph and clustering techniques. The algorithm is intended to reduce the number of passes over the database, and reduces the size of the candidate itemsets set. Obvious advantages of GCAR are the reduction of data scans that leads to I/O reduction and the reduction in the number of candidate itemsets that lead to CPU time saving. The rest of the paper is organized as follows. Section 2 provides background and related work. Our GCAR algorithm is introduced in Section 3. Section 4 presents experimental results. Finally, we conclude in Section 5 by summarizing our contributions and giving hints for future work.

2. Related Work

Association rules algorithms have became a widely researched area since their introduction in 1993 by Agrawal et al. [1]. The authors introduced what so called AIS. This algorithm makes multiple passes over the database. Candidate itemsets are generated and counted on-the-fly as the database is scanned. In each pass, new

candidate itemsets, that need to be counted, are 1-extensions (an itemset extended by exactly one item) of frequent itemsets found in the previous pass. The candidates generated from a transaction are added to the set of candidate itemsets, or their counts are increased if they were created by an earlier transaction. The AIS algorithm has several shortcomings: it makes many passes over the database and generates and counts too many candidate itemsets that turn out to be small. In addition, it is limited to only one item in the consequent part of the rules.

Apriori algorithm introduce in [2] is the most well known algorithm that makes multiple passes over the database to discover frequent itemsets. It uses the frequent itemsets in previous pass to generate candidate itemsets for next pass. At pass *k*, it generates the candidate *k*-itemsets from frequent *k*-1-itemsets and then scans the database to find the support of each candidate *k*-itemset. Itemsets with support above the minimum support form the frequent *k*-itemsets set. Other itemsets are discarded. The process continues until no more candidate itemsets can be found.

Generating candidate in Apriori itemsets is a two-step process: join step and prune step. In join step, two different *k*-1-itemsets are joined to generate *k*-itemset if their first *k*-2 items are in common. In prune step, all candidate *k*-itemsets, that have *k*-1-subset that is not frequent *k*-1-itemsets, are deleted. The prune step in candidate generation takes benefit of Apriori property that guarantees that all nonempty subsets of frequent itemset must also be frequent.

Itemsets are stored in hash tree. A node of the hash tree either contains a list of itemsets (a leaf node) or a hash table (an interior node). In an interior node, each bucket of the hash table points to another node. The root of the hash tree is defined to be at depth 1. An interior node at depth *d* points to nodes at depth *d*+1. When an itemset *c* is added, the search starts from the root and goes down the tree until a leaf is

reached. At an interior node at depth *d*, a hash function to the *d*-th item of the itemset is applied to decide which branch to be followed. All nodes are initially created as leaf nodes. When the number of itemsets in a leaf node exceeds a specified threshold, the leaf node is converted to an interior node.

Partition algorithm proposed in [3] is an efficient algorithm for disk resident databases because it scans the database twice to generate all significant association rules. An effective hash-based algorithm DHP (Direct Hashing and Pruning) that improves the overall performance proposed in [4]. The algorithm reduces the number of candidate *k*-itemsets, for $k > 1$. Additionally, Tovionen in [5] proposed sampling algorithm to produce exact association rules by not more than two full scans of database. Brin et al. in [6] proposed a dynamic itemset counting algorithm (called DIC) that finds frequent itemsets by fewer passes over the data than classic algorithms, and yet uses fewer candidate itemsets than other methods based on sampling. New algorithms that allow user to specify multiple minimum supports are proposed in [7, 8]. FP-growth algorithm that mines the complete set of frequent itemsets without candidate generation was proposed in [9]. The graph-based approach proposed in [10] scans the database only once to generate all frequent itemsets. It minimizes the time needed, while increases the required memory.

In [11] an algorithm named CBAR was proposed. The CBAR method creates cluster tables by scanning the database once, and then clustering the transactions to a k^{th} cluster, where the length of a record is *k*. The frequent itemsets are generated by scanning only partial cluster tables. This algorithm prunes great amount of data reducing the time needed to perform data scan. Additionally, CDAR [12] is another algorithm that combines clustering with decomposition of larger candidate itemsets down to frequent 1-itemsets. Firstly, CDAR likes CBAR creates cluster tables by

scanning the database once, and then clustering the transactions to a k^{th} cluster, where the length of a record is *k*. Then, the largest *k*-itemsets are generated by scanning the k^{th} cluster only, unlike combination of smaller itemsets that scans the entire database.

The Apriori algorithm is a widely used algorithm to mine association rules. It aims at finding all the associations among items, i.e., given a set of transactions, a specified minimum support, and a specified minimum confidence, it tries to find all the association rules satisfying both minimum support and minimum confidence.

3. Graph-Clustering Algorithm

Our Graph-Clustering Association Rules algorithm is an optimization of Apriori [2]. The proposed algorithm employs both graph and clustering techniques during the mining process in order to discover association rules. As described in Section 2, Apriori constructs a candidate set of itemsets, and then scans the database many times to discover indeed frequent itemsets. Our algorithm GCAR is more intelligent than Apriori in sense that it reduces the number of database scans by using graph-based technique. Moreover, it discards some infrequent candidate itemsets using a cluster-based technique.

The proposed algorithm, shown in Figure 1, has two main advantages: one is the reduction of database scans and the other is elimination of candidate *k*-itemsets of order 3 and above. The algorithm scans the database once to build a graph of items and a clustering table. This scan is enough to find frequent 1-itemsets and frequent 2-itemsets. As described in Section 3.1, no need to generate candidate 2-itemsets and hence no need to scan the database to discover frequent 2-itemsets. After that, GCAR works iteratively starting from frequent itemsets set F_2 in the sense that frequent

itemsets that are discovered in iteration *i* will be used as the basis to generate candidate itemsets in iteration *i*+1. The candidate generation step (described in Section 3.2) is similar to that in Apriori algorithm but here we employ clustering technique to eliminate some infrequent candidate itemsets.

```
(1)   GCAR(int min-sup)
(2)   G ← ∅
(3)   C1 = {set of all items}
(4)   for all transactions t ∈ D do
(5)      Build-Graph(G, t);
(6)      Cluster(t, length(t));
(7)   GraphFrequent(G);
(8)   for (k = 3; Fk-1 ≠ Φ; k++) do
(9)      Ck = Gen-Candidate(Fk-1);
(10)     for all transactions t ∈ D do
(11)        Ct = subset(Ck, t);
(12)           for all candidate c ∈ Ct do
(13)              c.count++;
(14)     Fk = {c ∈ Ck | c.count ≥ min-sup}
(15)  return ⋃k Fk

(1)   Build-Graph(graph G, transaction t)
(2)   for each item i ∈ t do
(3)      V[G] ← V[G] ∪ {i}
(4)      count[i] = 1;
(5)   for each 2-subset itemset e ∈ t do
(6)      E[G] ← E[G] ∪ {e}
(7)      count[e] = 1;
(8)   if (there are similar vertices and edges) then
(9)      Merge(vertices and edges);

(1)   Cluster(transaction t, int n)
(2)   for each item i ∈ t do
(3)      Clustering-table[i][n]++;

(1)   GraphFrequent(graph G)
(2)   for each vertex v ∈ V[G] do
(3)      if (count[v] ≥ min-sup) then
(4)         F1 ← F1 ∪ {v};
(5)   for each edge e ∈ E[G] do
(6)      if (count[e] ≥ min-sup) then
(7)         F2 ← F2 ∪ {e};
```

Figure 1: GCAR algorithm pseudocode.

3.1 Building GCAR Graph

The Build-Graph algorithm builds a complete undirected graph $G = (V, E)$ using all transaction in the database. Initially, the graph G is the subgraph of the first transaction. For each transaction t in the database, the algorithm builds a complete undirected subgraph $G_t = (V_t, E_t)$, where V_t is the set of all items in t and E_t is the set of all edges between every 2-subset itemsets in t. Associated with each vertex and edge is a counter that stores the occurrences of that vertex or edge and are initialized to 1. After building the subgraph G_t, a new version of graph G is created by merging G and G_t. If there are any similar vertices and edges between G_t and G, their counters are summed up. This process continues using all the transactions in the dataset. Figure 2 shows an example of three transactions and their subgraphs.

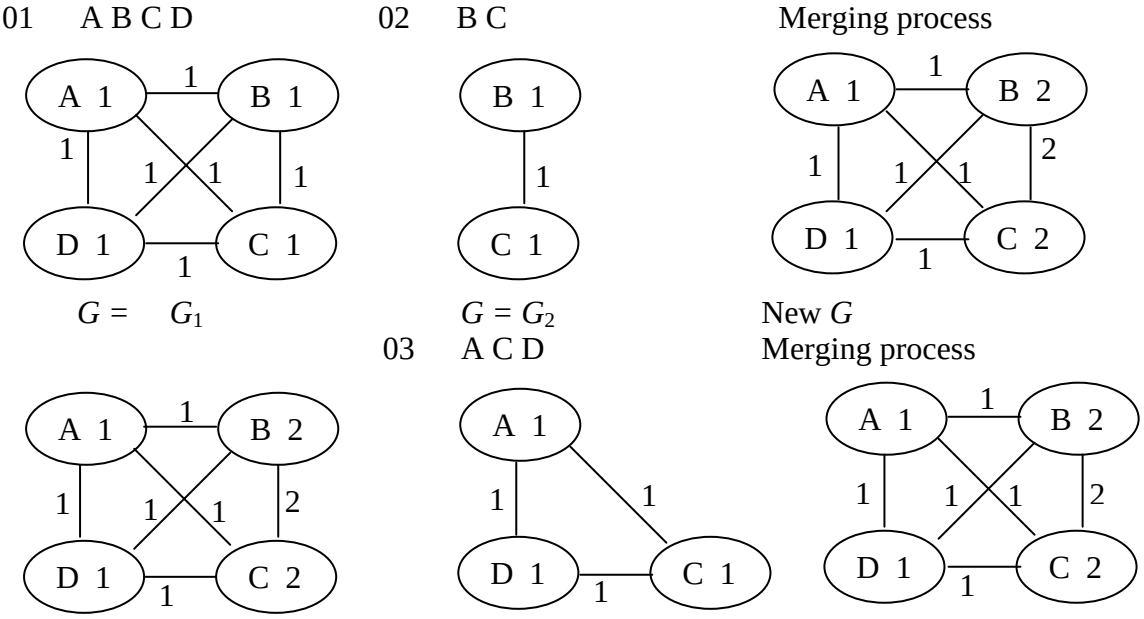

Figure 2: An example of three transcations and their sub graphs.

Vertices' counters hold the supports of the corresponding items. Edges' counters on the other hand hold the supports of the corresponding 2-subset itemsets. Finally, the function *Graph-Frequent* searches the graph to find frequent 1-itemsets and frequent 2-itemsets. The function *Graph-Frequent* traverses each vertex and edge in the graph, if the counter of a vertex is greater than or equal the minimum support

then the corresponding item is inserted into the set of frequent 1-itemsets F_1 and if the counter of a edge is greater than or equal the minimum support then the corresponding 2-subset itemset is inserted into the set of frequent 2-itemsets F_2.

3.2 Candidate itemsets generation using clustering

In the clustering step, each transaction is clustered to the k^{th} cluster, where the length of a transaction is *k*. Meanwhile, a clustering table is built to count the occurrences of each item in each cluster. Figure 3 shows the clustering-table through reading the first three transactions. Subsequently, clustering-table will be used in the candidate generation.

	01	A B C D					02	A B					03	A C D		
	C_1	**C_2**	**C_3**	**C_4**			**C_1**	**C_2**	**C_3**	**C_4**			**C_1**	**C_2**	**C_3**	**C_4**
A	0	0	0	1		A	0	1	0	1		A	0	1	1	1
B	0	0	0	1		B	0	1	0	1		B	0	1	0	1
C	0	0	0	1		C	0	0	0	1		C	0	0	1	1
D	0	0	0	1		D	0	0	0	1		D	0	0	1	1

Figure 3: An example of three transactions and their clustering-table

3.3 Candidate Generation

The *Gen-Candidate* function takes as argument F_{k-1}, the set of frequent *k*-1-itemsets, and returns the set of all candidate *k*-itemsets. The function performs two steps, namely, *join* step and *prune* step (see Figure 4). In the join step, two different *k*-1-itemsets are joined to generate *k*-itemset if their first *k*-2 items are common. In the prune step, two tests are performed. In the first test, all candidate itemsets, that have *k*-1-subset that is not in frequent *k*-1-itemsets, are deleted. In the second test, all

candidate itemsets, that have an item such that the sum of occurrences of that item in cluster *k* to cluster *m* is less than the minimum support, are deleted.

```
(1)   Gen-Candidate(frequent set F_{k-1})
(2)   for all itemset p ∈ F_{k-1} do
(3)     for all itemset q ∈ F_{k-1} do
(4)       if (p[1]=q[1], …, p[k-2]=q[k-2],p[k-1]<q[k-1]) then
(5)         insert into C_k (p[1], …, p[k-1],q[k-1]);
(6)         if (Not-Prune(c, F_{k-1})) then
(7)           add c to C_k;
(8)         else
(9)           delete c;

(1)   Not-Prune(candidate itemset c, frequent set F_{k-1})
(2)   for all k-1-subset s ∈ c do
(3)      if (s ∈ F_{k-1}) then
(4)         return false;
(5)   for all item a ∈ c do
(6)      if ( Test-Cluster(a, k) < minsup) then
(7)         return false;
(8)   return true;

(1)   Test-Cluster(item a, int k)
(2)   sum = 0;
(3)   for i = k to m do
(4)      sum = sum + clustering-table[a][i];
(5)   return sum;
```

Figure 4: Gen-Candidate algorithm pseudocode.

4. Experiments and Results

This section introduces a comparison between our proposed GCAR algorithm and Apriori algorithm using different synthetic and real datasets. The experiments were run on Pentium *M* computer with a clock rate of 1600 MHz and 256 Mbytes of main memory.

4.1 Synthetic Data

We used three synthetic datasets that were generated as described in [2]. These synthetic datasets are widely used for evaluating association rules algorithms [2, 4, 5,

6]. Table 1 shows the names and descriptions of parameters used to generate the different datasets. For the four datasets used in the experiments N was set to 1000 and $|L|$ was set to 2000.

Table 1: Synthetic data parameters.

$\|D\|$	Number of transactions
$\|T\|$	Average size of transactions
$\|I\|$	Average size of maximal potentially large itemsets
$\|L\|$	Number of maximal potentially large itemsets
N	Number of items

Figures 5 and 6 show the execution time for the first two synthetic datasets T10I4D100K and T20I10D100K, respectively. It is easy to see that GCAR beats Apriori for all minimum supports.

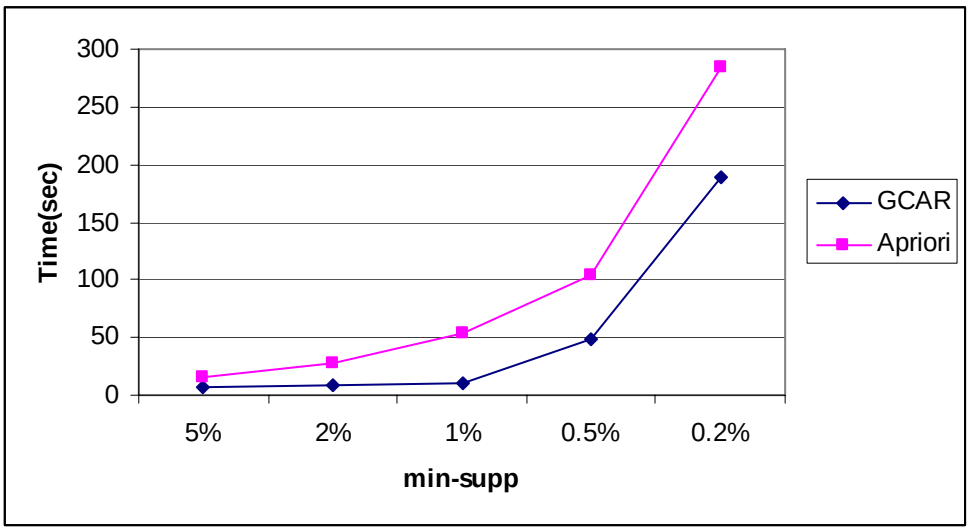

Figure 5: Execution time using (T10I4D100K) synthetic dataset

Figure 7 shows the execution time using the synthetic dataset T40I10D100K. From the figure, one can see Apriori algorithm is slower than GCAR as the minimum support reduces. The reason behind this is that as minimum support decreases the number of frequent itemsets increase.

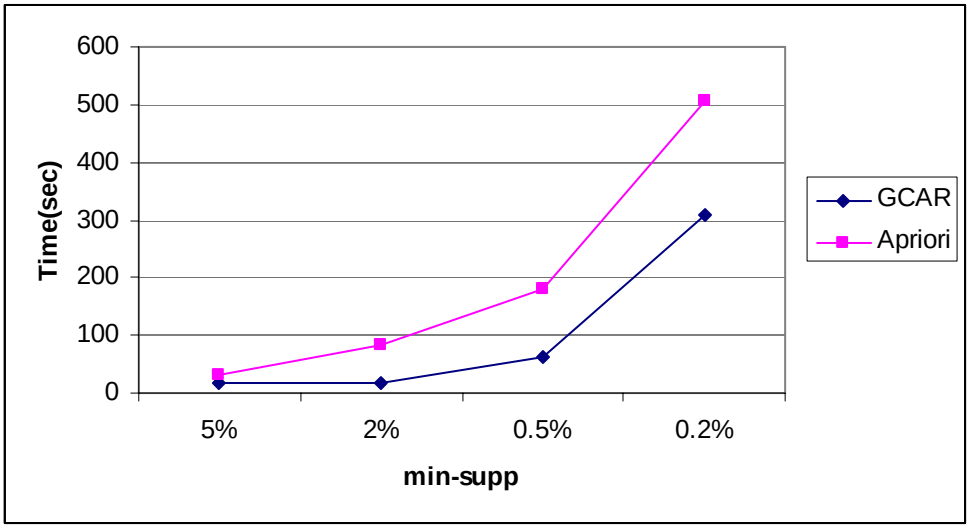

Figure 6: Execution time using (T20I10D100K) synthetic dataset

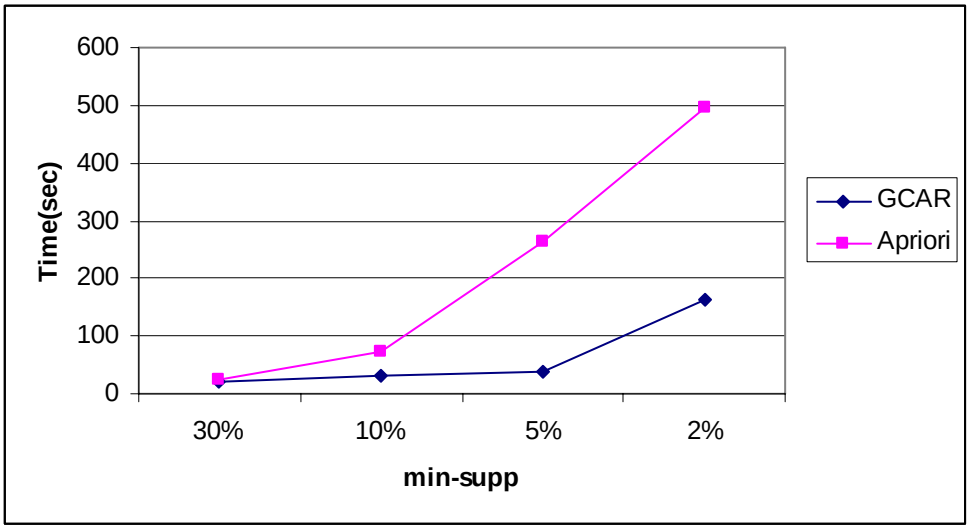

Figure 7: Execution time using (T40I10D100K) synthetic dataset

4.2 Real Data

We conducted several experiments on four real datasets: sales transactions database obtained from a retail database, web documents, mushroom database, and chess database [13].

Figure 8 shows the execution time for retail database. As we can see, the results are similar to that in Figure 5. GCAR beats Apriori, especially, when the minimum support is reduced. According to the results shown in Figures 9, 10, and 11, we can see that the execution time of GCAR is better than Apriori because GCAR

required less database scans than those required by Apriori and because GCAR employed clustering to delete some infrequent itemsets without being examined.

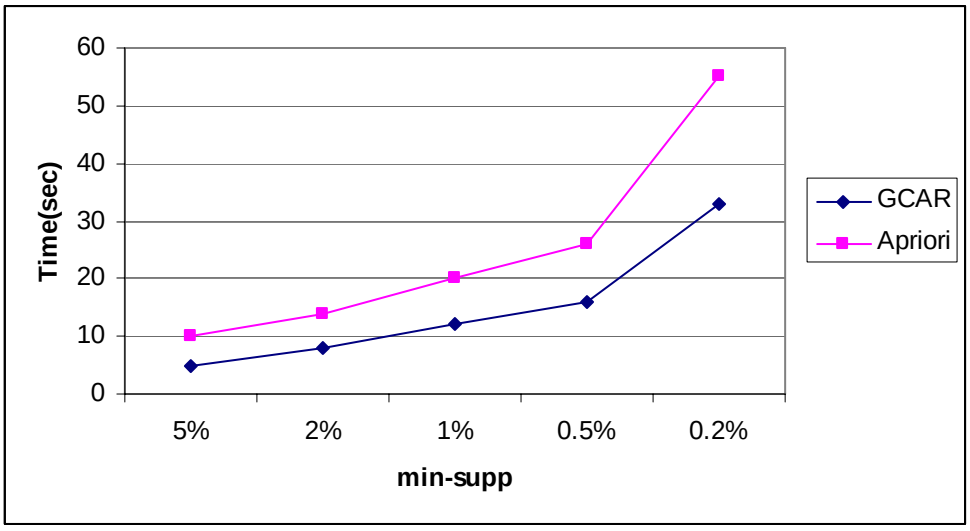

Figure 8: Execution time using (retail) dataset

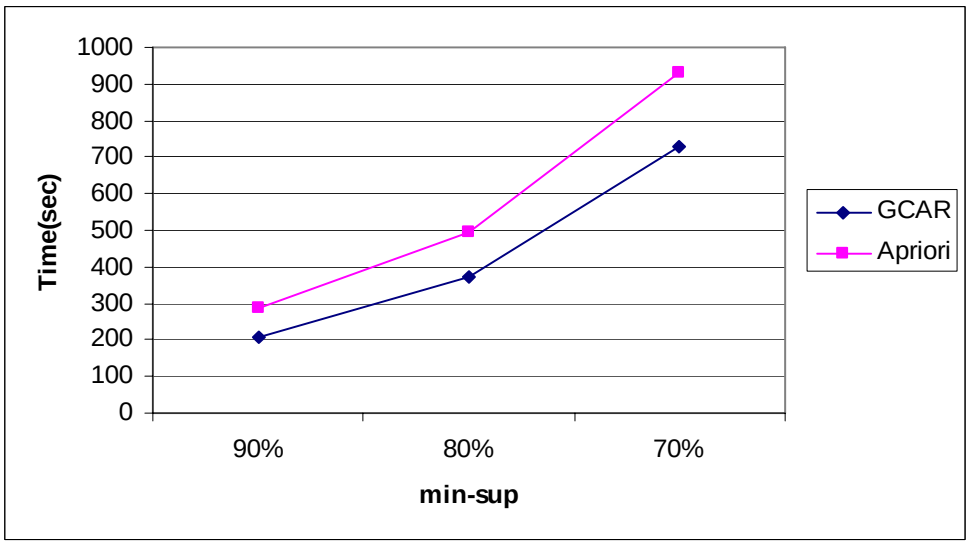

Figure 9: Execution time using (web document) dataset

5. Conclusions

In this paper, we presented an efficient algorithm for mining association rules. The proposed algorithm called GCAR is an enhancement of the Apriori algorithm. GCAR employs both graph and clustering techniques. From a practical point of view, the proposed algorithm is very attractive, since the graph technique reduces the database scans, while the clustering technique tries to eliminate some candidate

itemsets. We also performed an extensive experimental study using real, as well as synthetic data, which showed that the proposed algorithm outperforms Apriori algorithm, especially when the minimum support is reduced. In the future, we wish to compare GCAR with other association rule algorithms such as FP-growth.

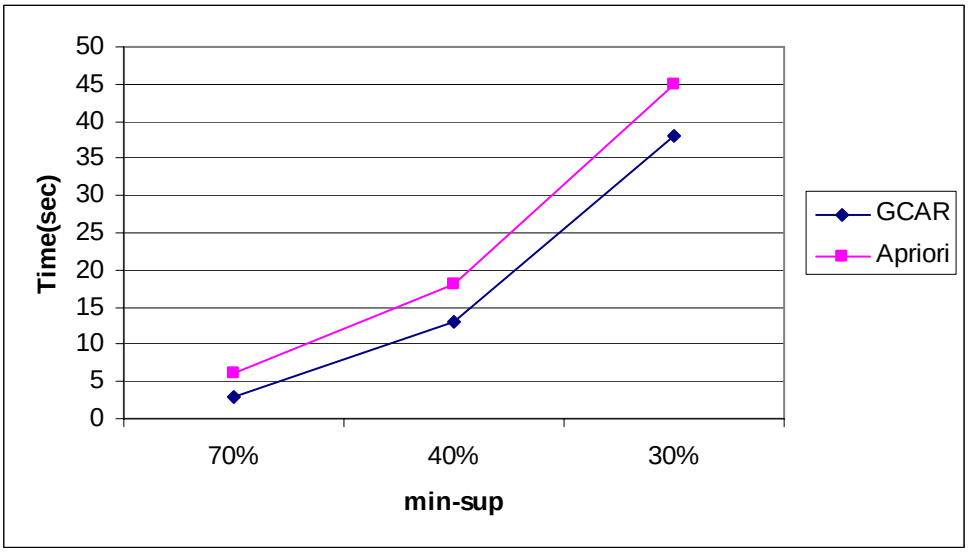

Figure 10: Execution time using (mushroom) dataset

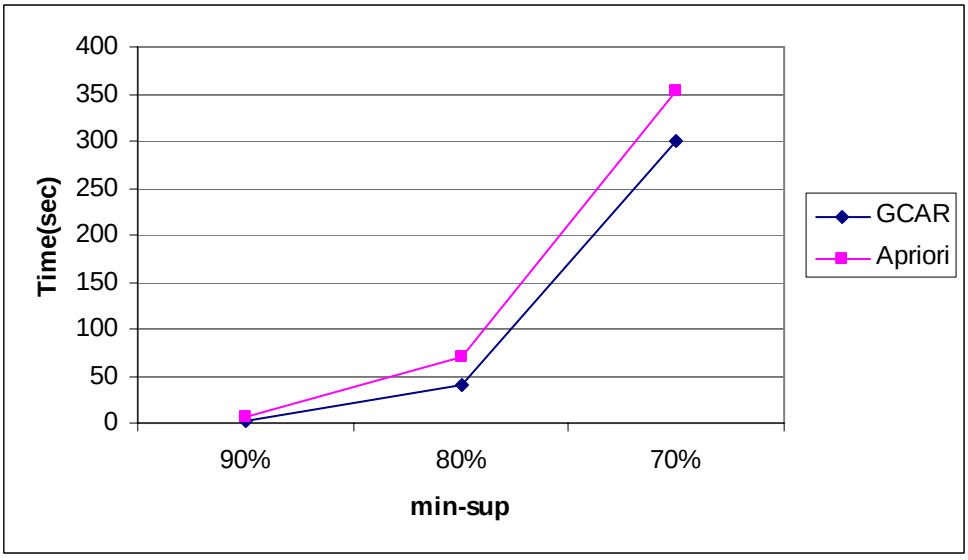

Figure 11: Execution time using (chess) dataset

References

[1] Agrawal R., Imilienski T., and Swami A., "Mining Association Rules Between Sets of Items in Large Databases", Proceedings of the ACM SIGMOD Conference, pp. 207-216, 1993.

[2] Agrawal R. and Srikant R., “Fast Algorithms for Mining Association Rules”, Proceedings of the 20th VLDB Conference, pp. 487-499, 1994.

[3] Savasere A., Omiecinski E., and Navathe S., “An Efficient Algorithm for Mining Association Rules in Large Databases”, Proceeding of the 21st VLDB Conference, pp. 432-444, 1995.

[4] Pork JS., Chen MS., and Yu PS., “An Effective Hash Based Algorithm for Mining Association Rules”, Proceedings of the ACM SIGMOD Conference, pp. 175-186, 1995.

[5] Toivonen H., “Sampling Large Databases for Association Rules”, Proceedings of the 22nd VLDB Conference, pp. 134-145, 1996.

[6] Brin S., Motwani R., Ullman JD., and Tsur S., “Dynamic Itemset Counting and Implication Rules for Market Basket Data”, Proceedings of the ACM SIGMOD Conference, pp. 255–264, 1997.

[7] Liu B., Hsu W., and Ma Y., “Mining Association Rules with Multiple Minimum Supports”, Proceedings of the 5th ACM SIGKDD Conference, pp. 337-341, 1999.

[8] Lee YC., Hong TP., and Lin WY., “Mining Association Rules with Multiple Minimum Supports Using Maximum Constraints”, International Journal of Approximate Reasoning, 40(1-2), pp. 44-45, 2005.

[9] Han J., Pei J., and Yin Y., “Mining Frequent Patterns without Candidate Generation”, Proceedings of the ACM SIGMOD Conference, pp. 1-12, 2000.

[10] Yen SJ. and Chen A., “A Graph-Based Approach for Discovering Various Types of Association Rules”, IEEE Transactions on Knowledge and Data Engineering, 13(5), pp. 839-845, 2001.

[11] Tsay YJ. and Chiang JY., “CBAR: An Efficient Method for Mining Association Rules”, Knowledge Based Systems, 18(2-3), pp. 1-7, 2004.

[12] Tsay YJ. and Chiang JY., “An Efficient Cluster and Decomposition Algorithm for Mining Association Rules”, Information Sciences; 160(1-4), pp. 161-171, 2004.

[13] Frequent Itemset Mining Dataset Repository, [accessed October 2005]. Available from URL http://fimi.cs.helsinki.fi/data/.