MINING FREQUENT ITEMSETS USING ADVANCED PARTITION APPROACH

by

KRANTHI K. MALREDDY, B.S.

A THESIS

IN

COMPUTER SCIENCE

Submitted to the Graduate Faculty
of Texas Tech University in
Partial Fulfillment of
the Requirements for
the Degree of

MASTER OF SCIENCE

Approved

[illegible]
Chairperson of the Committee

[illegible]

Accepted

[illegible]
Dean of the Graduate School

December, 2004

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor, Dr. Susan Mengel, for giving me an opportunity to work on this challenging topic and providing me ample guidance and support through the course of this research.

I would like to thank Dr. Yu Zhuang serving on my committee.

I also wish to express my deepest gratitude to Department of Ophthalmology and Visual Sciences, Texas Tech Health Sciences Center, whose financial help and support made my studies here at Texas Tech possible.

Next, I would like to deliver my sincere appreciation to Michael Holler, who provided us the implementation code of data mining algorithms and offer his expert help to this investigation and to Elizabeth Morris and Xie Xupeng for their useful discussion and assistance in countless ways.

I would like to thank Derek Birkenfeld for maintaining a well-administered research environment and being so helpful at times of need. I would like to thank all my friends in the Software Engineering Lab. I would also like to thank my friends for their support and encouragement.

I would also like to thank my parents and brother for their endless love and constant support throughout my academic career without which I would not have reached this position.

November 08, 2004

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Data Mining is the process of extracting interesting and previously unknown patterns and correlations from data stored in Database Management Systems (DBMSs). Association rule mining, a descriptive mining technique of data mining is the process of discovering items, which tend to occur together in transactions. As the data to be mined is large, the time taken for accessing data is considerable.

In this thesis, a new Association rule mining algorithm which generates the frequent itemsets in a single pass over the database is implemented. The algorithm mainly uses two approaches for association rule mining over data stored in multiple relations in one or more databases: The Partition approach, where the data is mined in partitions and merges the result, and the Apriori approach that helps to find the frequent sets within each partition. In order to evaluate the performance of the new association algorithm, it is compared with the existing algorithms which require multiple database passes to generate the frequent itemsets. Extensive experiments are performed and results are presented for both the approaches. Experiments show that time taken for the database scan is more than the time taken for the candidate generation when the database size is large, which provides evidence that focus to decrease the database access time is a viable approach to the association rule mining.

CHAPTER I

INTRODUCTION

Due to widespread computerization and affordable storage facilities, an enormous wealth of information is embedded in huge databases belonging to different enterprises. Such databases, whether their origin is the business enterprise or scientific experiment, have spurred a tremendous interest in the areas of Knowledge Discovery and Data Mining [ARUN2003] [MARG2003].These areas have motivated allowed statisticians and data miners to develop faster analysis tools that can help sift and analyze the stockpiles of data, turning up valuable and often surprising information.

Data mining is the act of drilling through huge volumes of data to discover relationships, or answer queries too generalized for traditional query tools. Data Mining is part of the process know as Knowledge Discovery in Databases (KDD) [ARUN2003] [MET2000], which is the automated approach of the extraction of implicit, understandable, previously unknown and potentially useful information from large databases. For extraction of such valuable information, the KDD process follows an iterative sequence of steps that include data selection and integration, data cleaning and preprocessing, data mining and algorithm selection, and, finally, post processing and knowledge presentation. Figure 1.1 illustrates a typical KDD process.

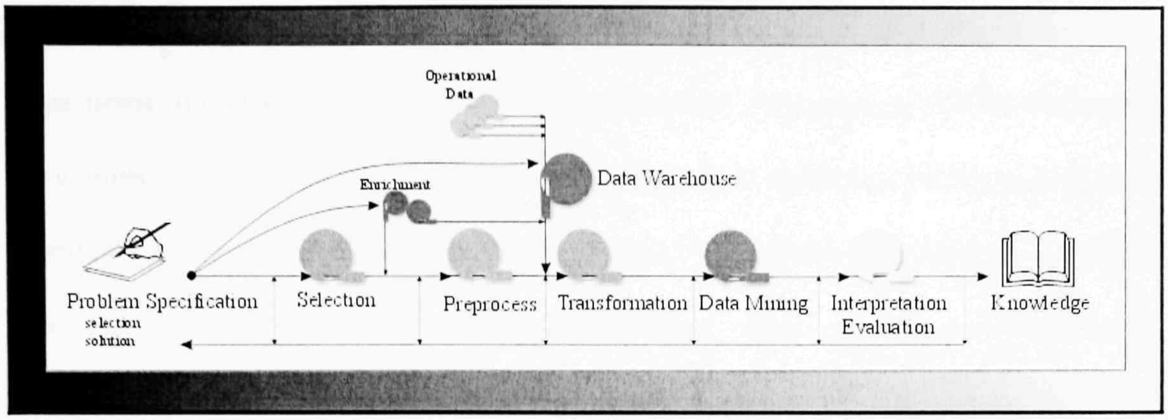


Figure 1.1. A Knowledge Discovery in Databases Process. Adapted from [MET2000].

In general, Data Mining tasks can be classified into two categories:

Descriptive Mining: This is the process of drawing patterns in existing data, and is generally used to create meaningful subgroups, such as demographic clusters. Clustering, Association Rule Mining, and Sequential Mining are some of the Descriptive Mining techniques [MET2000] [ARUN2003] [MARG2003].

Predictive Mining: This is used to forecast explicit values, based on patterns determined from known results. For example, from a database of customers who have already responded to a particular offer, a model can be built that predicts which prospects are likeliest to respond to the same offer. Classification, Regression, and Deviation detection are Predictive Mining techniques [MET2000] [ARUN2003] [MARG2003].

Among all the techniques listed above, Association Rule Mining [ARUN2003] [MARG2003] which is a Descriptive Mining technique has received a great deal of attention. This technique was formulated by Agarwal et al. in 1993 and is often referred to as the *market-basket problem* [ARUN2003]. In this market-basket problem, when given a set of items and a large collection of transactions, which are subsets (baskets) of those items, the relationships between the presences of various items within the baskets are explored. The way the market-basket problem gets its name is from the supermarket, shopping basket where customer's buying habits are analyzed by finding associations between the different items that customers purchase. Those associations can help the retailer develop marketing strategies and inventory management, by gaining insight into matters, such as "which items are most frequently purchased by customers"

1.1. Motivation of Thesis

Generation of association rules is solely dependent on the generation of frequent item sets. Thus, algorithms which are used to generate association rules are concerned with efficiently determining the set of frequent itemsets in a given set of transactions. In other words, determining the frequent itemsets means to compute the frequency of occurrences of each itemset in the database. Itemsets thus generated are exponential in terms of the number of items. So it's not possible to count the frequencies of these sets by reading the database in just one pass, so more than one pass is unavoidable for generating all the frequent itemsets. Thus, algorithms which are used to generate association rules aim at reducing the number of passes by generating candidate sets [ARUN2003] [MARG2003], which are likely to be frequent sets. These algorithms differ from one another in the method of handling the candidate sets and the method of reducing the number of database passes. In addition, algorithms handling candidate sets may have difficulty when underlying database is incremented intermittently because they may have to compute the frequent sets afresh for the incremented set of data.

Here is the example to illustrate the above frequent itemset concept, let these are the set of transactions in a bookshop:

t_1:= { ANN, CC, TC, CG }

t_2:= { CC, D, CG }

t_3:= { ANN, CC, TC, CG }

t_4:= { ANN, CC, D, CG }

t_5:= { ANN, CC, D, TC, CG }

t_6:= { CC, D, TC }

t_1, t_2, t_3, t_4, t_5, t_6 are the set of transactions.

CC, D, TC, CG and ANN are the codes of the books.

If user specified that, the set is frequent if it is present in at least 50% of the transactions, So for example ANN is the book where it is present in 4 out of 6 transactions, so this ANN is a frequent set. And also ANN and CC are together in 3 transactions out of 6, so here the frequent set is { ANN, CC }

Of the issues that need to be considered in order to make generation of frequent sets efficient, reducing the number of passes over the database is the key, because, for example in disk-resident databases, where large numbers of disk-reads are required for reading the database in each pass, time spent in performing just I/O may be considerable. If, the disk-resident database of 1 gigabyte requires 125,000 block reads for a single pass (for a block size of 8 KB) and if the algorithm requires 10 passes, 1,250,000 – block reads results. Assuming an average read time of 12 ms per page, the time spent in performing the I/O is 125,000 x 12 ms = 4 hours. Thus, to reduce the I/O operations and at the same time be efficient in computing is a promising area where new techniques are yet to be exploited.

1.2. Goal of Thesis

The goal of this thesis research is to design a new Association rule mining algorithm which generates the frequent itemsets in a single pass over the database. The algorithm mainly uses two approaches for association rule mining over data stored in multiple relations in one or more databases: The partition approach [AGG1998] [FAST1995] [AGS1996], where the data is mined in partitions and merges the result, and the Apriori mining approach [AGG1998] [FAST1995] [AGS1996] that helps to find frequent itemsets within each partition.

1.3. Thesis Organization

The rest of this thesis is organized as follows. The next chapter, Chapter 2, presents an overview of pertinent literature and research of this thesis including general issues and examples of association rule mining algorithms. Chapter 3 describes the design and implementation of the new association rule mining algorithm based on the partition approach. Chapter 4 shows the results and discussions on experiment with the algorithm. Conclusions and potential future research directions are presented in Chapter 5.

CHAPTER II

LITERATURE REVIEW

This chapter covers the literature relevant to the thesis research and is organized into four sections. Section 2.1 outlines some basic concepts and methods to discover association rules. Section 2.2 enumerates some common approaches for generating the frequent itemsets. Section 2.3 compares the different approaches which are discussed in the previous section. Section 2.4 summarizes the important issues and topics covered in this chapter.

2.1 Association Rule Mining: Basic Concepts and Methods

2.1.1. Basic Concepts

Association rule mining [ARUN2003] [MARG2003] is a descriptive data mining technique used for uncovering relations among data. For example, a typical application of association rule mining is market-basket analysis [ARUN2003] [MARG2003], where the buying habits of the customers are analyzed to find associations among the different items that customers place in their shopping baskets. These associations are often used in the retail sales community to identify items that are frequently purchased together in a transaction or frequent in a itemsets. The associations generated are the form of a rule, $X \Rightarrow Y$, where X and Y are the sets of items. The intuitive meaning of such a rule is that the transaction in a database which contains X tends to contain Y. When given a transaction database, the goal is to discover all of the rules and the frequent itemsets in that transaction database.

Association rule mining utilizes two measures to find rules and frequent itemsets, Support and Confidence.

The example below explains how support and confidence are computed for items and rules using the following set of transactions in a bookshop, where ANN, CC, D, TC, CG are book codes.

t_1:= { ANN, CC, TC, CG }

t_2:= { CC, D, CG }

t_3:= { ANN, CC, TC, CG }

t_4:= { ANN, CC, D, CG }

t_5:= { ANN, CC, D, TC, CG }

t_6:= { CC, D, TC }

In the above transactions, item D is supported by 4 out of 6 transactions in T making the support of D 66.6%.The confidence of ANN $\Rightarrow$ CC is 100%, since all of the transactions that support ANN also support CC. The support of ANN $\Rightarrow$ CC is 66.6%, as both ANN and CC occur together in 4 out of 6 transactions. On the other hand, the confidence of CC $\Rightarrow$ ANN is 66.6% and its support 66.6%. While support does not depend on the direction (or implication) of the rule, confidence does.

More formally, let B = $\{l_1, l_2, \ldots l_m\}$ be a set of items and T be a set of transactions, where each transaction t in T is a subset of B. A transaction t is said to support an item l_i, if l_i is present in t and t is said to support a subset of items $X \subseteq B$, if t supports each item l_i in X. In other words, an itemset $X \subseteq B$ has a support percentage in T, denoted by $s(X)_T$, for the percentage of transactions in T that support X. For association rules of the form $X \Rightarrow Y$, where X and Y are subsets of B, the rule $X \Rightarrow Y$ holds confidence τ which is the number of transactions in T supporting (X U Y) / number of transactions supporting X. The rule, $X \Rightarrow Y$ has support σ which is the number of transactions in T supporting (X U Y) / total number of transactions. The intuitive meaning of association rule support and confidence is that a transaction in the database which contains X tends to contain Y. Further, given a set of transactions, T, the problem of mining association rules is to discover all rules that have support and confidence greater than or equal to a user-specified minimum support and minimum confidence, respectively.

2.1.2 Methods to Discover Association Rules

Finding association rules can be decomposed into two sub tasks [ARUN2003]:

1. Discover all sets of items (itemsets) whose support is greater than the user-specified minimum support, σ. Such itemsets are called frequent itemsets.
2. Use the frequent itemsets to generate the desired rules with at least the specified minimum confidence. The general idea is that if, say ABCD and AB are frequent itemsets, then determine if the rule $AB \Rightarrow CD$ holds by checking the following inequality

$$\frac{s(\{A, B, C, D\})}{s(\{A, B\})} \geq \tau_{\min}$$

Where s(X) is the support of X in T.

Research has been focused on the first task, because databases on which frequent itemsets are mined are generally large (terabytes), causing time for I/O operations when finding frequent itemsets to become significant or higher than time for rule generation after frequent itemsets are found.

The types of sets found in step one are given below:

Frequent set

An itemset $X \subseteq A$ is said to be a frequent itemset in T with respect to σ, if $s(X)_T \geq \sigma_{min}$. To illustrate, the example in the previous section is used. If $\sigma_{min} = 50\%$, then itemset, {ANN, CC, TC}, is a frequent set as it is supported by at least 3 out of 6 transactions. Once itemset, {ANN, CC, TC}, is a frequent set, any subset of this set is also frequent set. On the other hand, itemset {ANN, CC, D} is not a frequent set and hence, no set which properly contains this set is a frequent set. Fortunately, frequent sets exhibit the following two properties:

- Downward Closure Property:

  Any subset of a frequent set is a frequent set.

- Upward Closure Property:

  Any superset of an infrequent set is an infrequent set.

These properties help to reduce the combinatorial search space of itemsets and improve efficiency.

2.2 Basic Association Rule Mining Algorithms

2.2.1 Apriori Algorithm

The Apriori algorithm [AGG1998] [FAST1995] [AGS1996] is also called the level-wise algorithm and was proposed by Agrawal and Srikanth in 1994. It is the most popular algorithm to find all of the frequent sets which uses the downward closure property. The advantage of the algorithm is that before reading the database at every level, it prunes many of the sets which are unlikely to be frequent sets by using the Apriori property, which states that all nonempty subsets of frequent sets must also be frequent. This property belongs to a special category of properties called anti-monotone in the sense that if a set cannot pass a test, all of its supersets will fail the same test as well.

Using the downward closure property and the Apriori property, this algorithm works as follows. The first pass of the algorithm counts the number of single item occurrences to determine the L_1 or single member frequent itemsets. Each subsequent pass, k, consists of two phases. First, the frequent itemsets L_{k-1} found in the (k-1)th pass are used to generate the candidate itemsets C_k, using the Apriori candidate generation algorithm described below. Next, the database is scanned and the support of the candidates in C_k is determined to ensure that C_k itemsets are frequent itemsets.

Candidate Generation Algorithm

The candidate generation procedure works as follows. Suppose that the set of frequent 3-itemsets, L_3 , are {1, 2, 3}, {1, 2, 5}, {1, 3, 5}, {2, 3, 5}, {2, 3, 4}. The 4-itemsets that are generated as candidate itemsets are the supersets of these 3-itemsets and are {1, 2, 3, 5}, {2, 3, 4, 5}, which satisfy the downward closure property. More formally, if k is the pass number, L_{k-1} is the set of all frequent (k-1)-itemsets, C_k is the set of candidate sets of pass k, the candidate generation procedure is as follows:

```
gen_candiadate_itemsets with the given $L_{k-1}$ as follows:
 $C_k = \Phi$
 for all itemsets $l_1 \in L_{k-1}$ do
 for all itemsets $l_2 \in L_{k-1}$ do
     if $l_1[1] = l_2[1] \wedge l_1[2] = l_2[2] \wedge \ldots \wedge l_1[k-1] < l_2[k-1]$
     then $c = l_1[1], l_1[2] \ldots l_1[k-1], l_2[k-1]$
     $C_k = C_k \cup \{c\}$
```

So once candidate sets are generated those sets are subject pruning process to ensure that all the subsets of the candidate set are already known to be frequent itemsets.

Pruning Algorithm

The pruning step eliminates some candidate sets which are not found to be frequent, and is:

```
 prune($C_k$)
 for all $c \in C_k$
 for all (k-1)- subsets d of c do
   if d $\notin L_{k-1}$
   then $C_k = C_k - \{c\}$
```

Apriori Algorithm Description

The Apriori frequent itemset discovery algorithm uses the above algorithms (candidate generation and pruning) at every iteration. It goes from level 1 to level k or until no candidate set remains after pruning. The Apriori algorithm is as follows.

```
Initialize: k:= 1, $C_1$ = all the 1 – itemsets;
read the database to count the support of $C_1$ to determine $L_1$.
$L_1$ := {frequent 1-itemsets};
k := 2; // k represents the pass number //
while ( k-1 $\neq$ Null set) do
begin
  $C_k$ := gen_candidate_itemsets with the given $L_{k-1}$
  prune($C_k$)
  for all transactions t $\in$ T do
  Calculate the support values;
  $L_k$ := All candidates in $C_k$ with a minimum support;
  k := k + 1;
end
Answer : = $\cup_k L_k$;
```

For an example of the Apriori algorithm, suppose the following transaction database is given below:

A_1	A_2	A_3	A_4	A_5

t_1	1	0	0	0	1
t_2	0	1	0	1	0
t_3	0	0	0	1	1
t_4	0	1	1	0	0
t_5	0	0	0	0	1

Suppose $\sigma_{\min}$= 20%, which means that an itemset must supported by at least one transaction to be frequent because T only has five records.

In the first pass, where k= 1, T is read to find the support of the 1-itemsets given below.

$\{1\} \rightarrow 1, \{2\} \rightarrow 2, \{3\} \rightarrow 1, \{4\} \rightarrow 2, \{5\} \rightarrow 3$

$L_1 := \{ \{1\} \rightarrow 1, \{2\} \rightarrow 2, \{3\} \rightarrow 1, \{4\} \rightarrow 2, \{5\} \rightarrow 3 \}$

In the second pass where k = 2, the candidate set C_2 becomes

$C_2 := \{ \{1,2\}, \{1,3\}, \{1,4\}, \{1,5\}, \{2,3\}, \{2,4\}, \{2,5\}, \{3,4\}, \{3,5\}, \{4,5\} \}$

The pruning step does not change C_2 as all subsets are present in C_1.

Read the database to count the support of elements in C_2 to get:

$\{ \{1,2\} \rightarrow 0, \{1,3\} \rightarrow 0, \{1,4\} \rightarrow 0, \{1,5\} \rightarrow 1, \{2,3\} \rightarrow 1, \{2,4\} \rightarrow 1, \{2,5\} \rightarrow 0, \{3,4\} \rightarrow 0, \{3,5\} \rightarrow 0, \{4,5\} \rightarrow 1 \}$ and reduces to

$L_2 = \{ \{1,5\} \rightarrow 1, \{2,3\} \rightarrow 1, \{2,4\} \rightarrow 1, \{4,5\} \rightarrow 1 \}$

In the third pass where k = 3, the candidate generation step proceeds by:

In the candidate generation step,

- Using {1,5} and {4,5} it generates {1, 4, 5}
- Using {2,3} and {2,4} it generates {2, 3, 4}

- Using {2,4} and {4,5} it generates {2,4,5}

So C_3 := { {1,4,5}, {2,3,4, {2,4,5}}

The pruning step prunes {1, 4, 5}, {2, 3, 4}, {2, 4, 5} as not all subsets of size 2, i.e., {1, 4}, {3,4}, {2,5} are not present in L_3.

So $C_3 := \Phi$

The total frequent sets become $L: = L_1 \cup L_2$.

2.2.2 Partition Algorithm

The partition algorithm [AGG1998] [FAST1995] [AGS1996] is based in the observation that the frequent sets are normally very few in number compared to the set of all itemsets. As the result, if the set of transactions are partitioned into smaller segments such that each segment can be accommodated in the main memory, then the set of frequent sets of each of these partitions can be computed. Therefore this way of finding the frequent sets by partitioning the database may improve the performance of finding large itemsets in several ways:

- By taking advantage of the large itemset property, this is that a large itemset must be large in at least one of the partitions. This idea can help to design algorithms more efficiently than those based on looking at the entire database.
- Partitioning algorithms may be able to adapt better to limited main memory. Each partition can be created such that it fits into main memory. In addition it would be

expected that the number of itemsets to be counted per partition would be smaller than those needed for the entire database.

- By using partitioning, parallel and/or distributed algorithms can be easily created, where each partitioning could be handled by a separate machine.
- Incremental generation of association rules may be easier to perform by treating the current state of the database as one partition and treating the new entries as a second partition.

In order to achieve all the above advantages of partitioning the transaction database, the partition algorithm works as follow:

The partition algorithm uses two scans of the database to discover all frequent sets. In one scan, it generates a set of all potential frequent itemsets by scanning the database. This set is a superset of all frequent itemsets, i.e. it may contain false positives, but no false negatives are reported. During the second scan, counters for each of these itemsets are setup and their actual support is measured in one scan of the database.

The partition approach of generating frequent itemsets is given below:

```
P = partition_database(T); N = Number of partitions;

// Phase I
  for i = 1 to n do begin
  read_in_partition( $T_i$ in P )
  $L_i$ = generate all frequent itemsets of $T_i$ using a priori method in main
memory.
```

```
end

// Merge Phase
  for ( k = 2 ; $L_i^k \neq \Phi$ , i = 1,2,...,n ; k++) do begin
      $C_k^G = \bigcup_{i=1}^{n} L_i^k$
  end

// Phase II
  for i = 1 to n do begin
      read_in_partition( $T_i$ in P)
      for all candidates $c \in C^G$ compute $s(c)_{Ti}$
  end
  $L^G = \{ c \in C^G / s(c)_{Ti} \geq \sigma \}$

  Answer = $L^G$
```

As given the partition algorithm above, here is the example of implementing it:

	A_1	A_2	A_3	A_4	A_5
T_1	1	0	0	0	1
T_2	0	1	0	1	0
T_3	0	0	0	1	1
T_4	0	1	1	0	0

T_5	0	0	0	0	1
T_6	0	1	1	1	0

Here is the transaction database, $A = \{A_1, A_2, A_3, A_4, A_5\}$, assume $\sigma = 20\%$.

Here the database is partitioned into 3 partitions say T_1, T_2, T_3, each containing 2 transactions. The first partition T_1 contains 1 to 2 transactions, T_2 contains 3 to 4, and T_3 contains 5 to 6 transactions. Here the local support is equal to the given support, which is 20%. So $\sigma = \sigma_1 = \sigma_2 = \sigma_3 = 20\%$.

The working of partition algorithm is as follows:

L_1 := the frequent sets from the partition in T_1, which are found using the a priori algorithm on T_1 separately.

L_2 := the frequent sets from the partition in T_2, which are found using the a priori algorithm on T_2 separately.

L_3 := the frequent sets from the partition in T_3, which are found using the a priori algorithm on T_3 separately.

In phase II, the candidate set as

$C := L_1 \cup L_2 \cup L_3$

Later read the database once again to compute the global support of the sets in C and get the final set of frequent sets.

2.3 Discussion on Different Algorithms

Of the above algorithms, the two important and common steps are candidate generating and I/O operations. The initial candidate set generation, especially for the frequent 2-itemsets, is the key issue to improve performance of the frequent set discovery algorithms. Another performance measure is the amount of data that has to be scanned during the discovery of the frequent itemset.

So to measure the performance of algorithms, the Apriori and the Partition, which handle static databases, datasets presented in FIMI repository for frequent itemset mining (http://fimi.cs.helsinki.fi/data/) have been used. The nomenclature of these datasets is of the form "TxxDzzzK", where "xx" denotes the average number of items present per transaction and "zzzK" denotes the total number of transaction on "K"(1000's). The experiments have been performed on a machine running Microsoft Windows XP with 786 MB of RAM and 1.6 Ghz Processor. Each experiment has been performed 4 times. The values from the first run are ignored so as to avoid the effect of the previous experiment and other database setups. The average of the next 3 runs is taken and used for analysis. This is done so as to avoid any false reporting of time due to system overload or any other factors. For most of the experiments, the percentage of difference of each run is less than one percent.

For performance analysis the datasets chosen are as follows:

Name	|T|	|D|	Size in Megabytes
T9D80K	9	80K	3.5
T13D90K	13	80K	4
T13D180K	13	350K	8
T13D360K	13	350K	16

Table 2.1 : Parameter settings

Firstly to compare the performance of both the algorithms the way they handle the aspect of candidate generation, the two sample datasets (T9D80K and T13D80K) of same size (80K transactions) and having different average number of itemsets per transaction are taken. Below figures 1 and 2 shows the execution times of Apriori and Partition algorithm under different minimum supports.

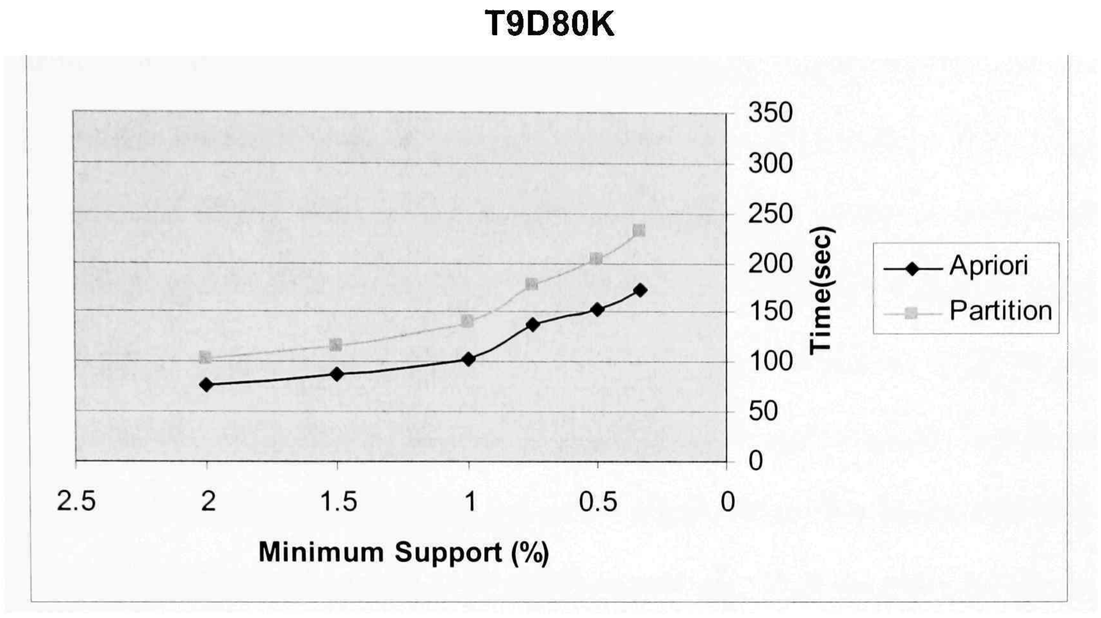


Figure 2.1 Performance T9D80K

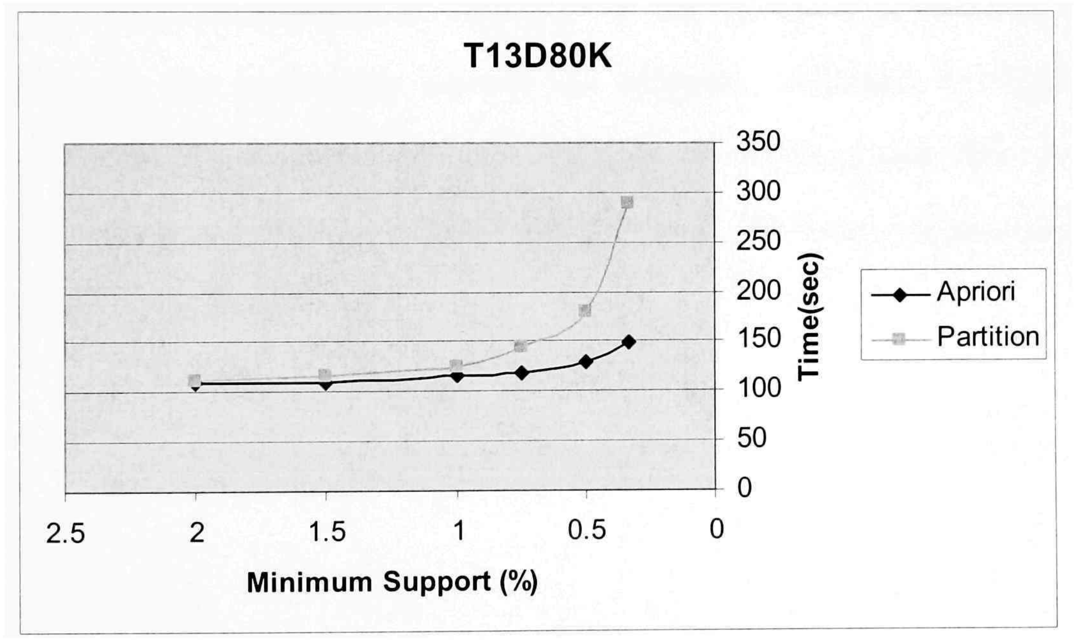


Figure 2.2 Performance T13D80K

Both the above figures show that, as the minimum support decreases, the execution times of both the algorithms increase because of increase in the total number of candidate and large itemsets. Figure 2 (dataset with more number of average number of itemsets per transaction, in other words, more number of candidate sets are generated) shows that execution times of Apriori algorithm beats the execution time of partition algorithm by almost factor of two as the minimum support decreases (more number of candidate sets are generated as support decreases). But in Figure 1, where less number of candidate sets are generated, shows that execution times of Apriori algorithm beats the execution times of partition algorithm but difference of execution time is same even the candidate sets increase (as support decreases). So this shows that Apriori algorithm much efficiently handles the generation of candidate generation than the partition algorithm.

To measure how both the algorithms handle the I/O operations, the execution times of both the algorithms are measured with different sizes of datasets. The datasets taken for this performance measure are T13D80K, T13D90K, T13D180K, and T13D360K. Figure 3, Figure 4, Figure 5, Figure 6 shows the execution times of Apriori and partition algorithm for different database sizes and having minimum percentage support 2, 4, 6, 8 respectively.

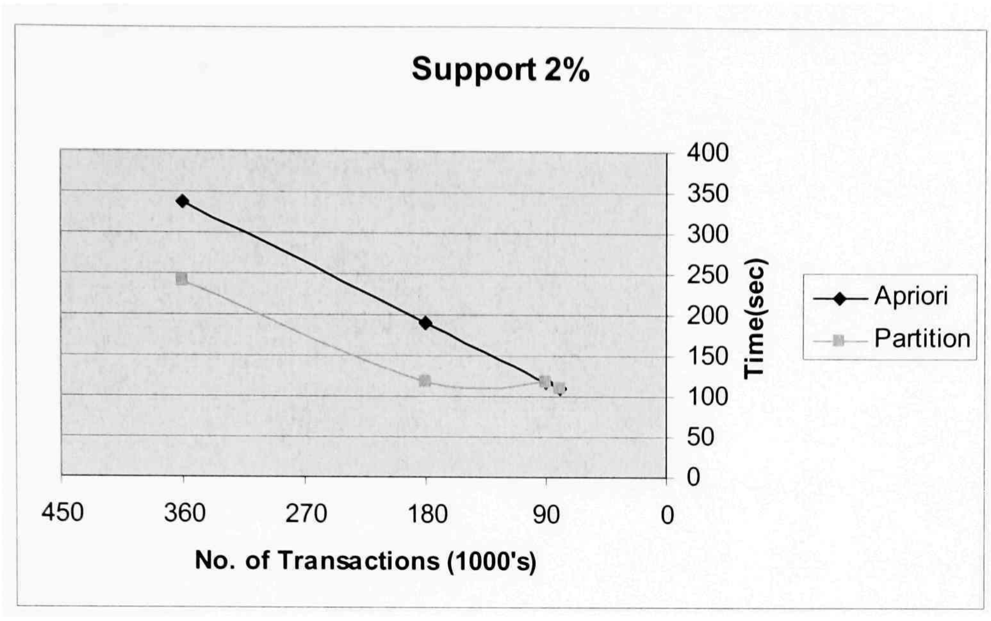


Figure 2.3 Performance Support 2%

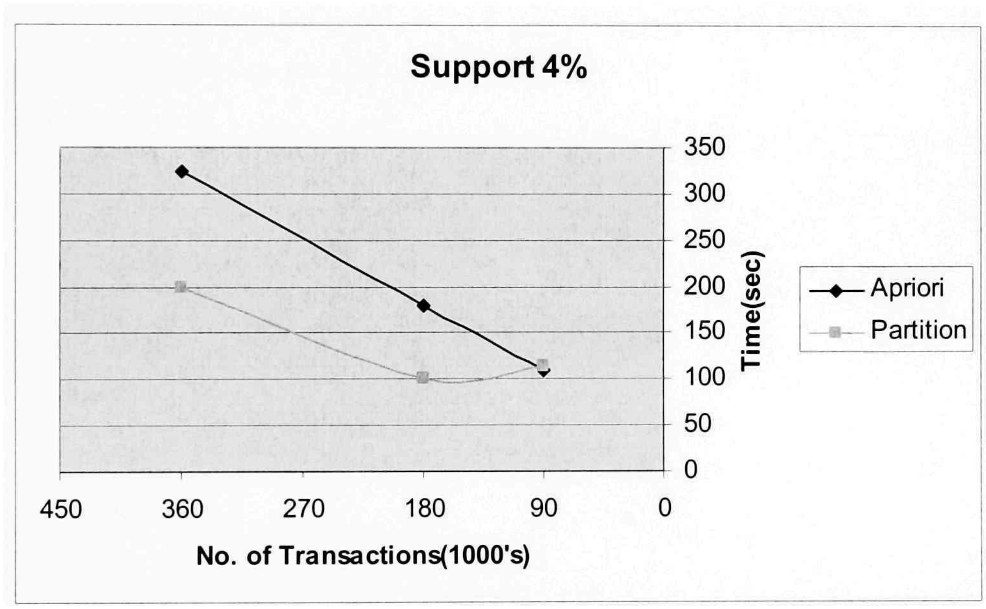


Figure 2.4 Performance Support 4%

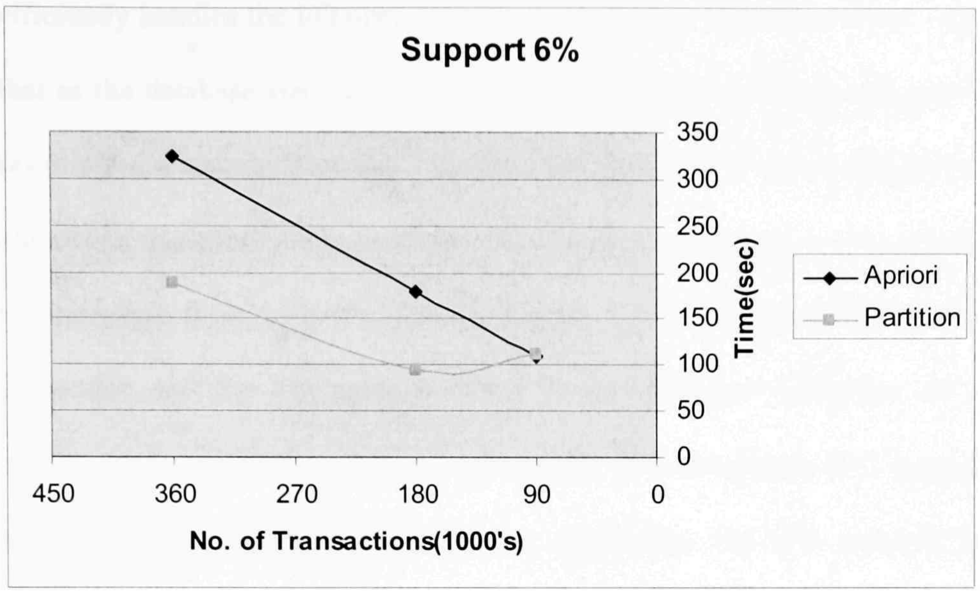


Figure 2.5 Performance Support 6%

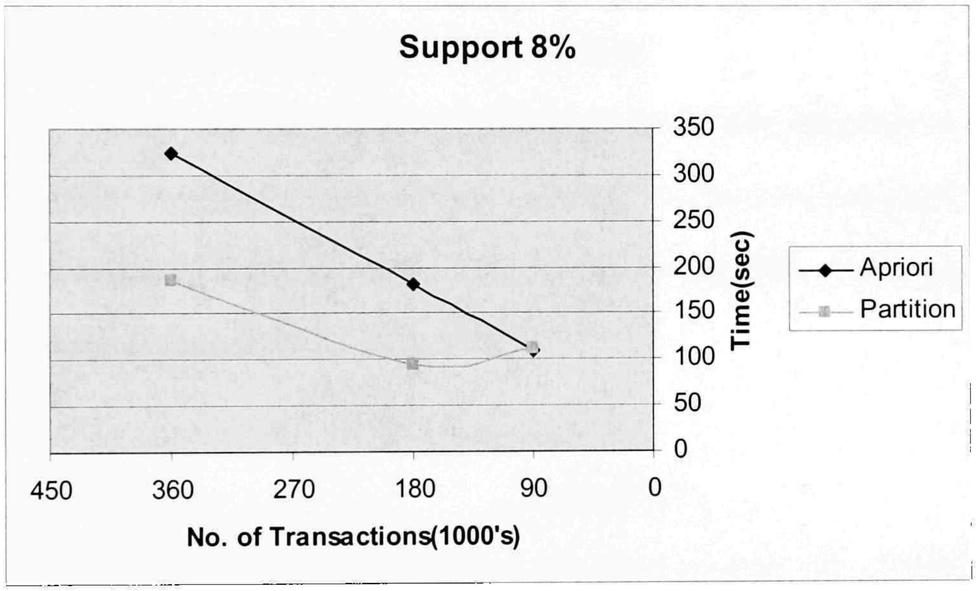


Figure 2.6 Performance Support 8%

From the above figures it shows that Partition algorithm beats the execution time of Apriori algorithm as database size increases. In other words, Partition algorithm

efficiently handles the I/O operations than the Apriori algorithm. Those figures also show that as the database size increases and the number of candidate sets generated decreases (as minimum support decreases) the partition algorithm is more efficient than the Apriori algorithm(as difference in execution times increase as the support increase).
On the whole from the performance measures of both the algorithms on the candidate set generation and the I/O operations, the most significant difference of execution time comes when increase or decrease of number of transactions (I/O operations) than the increase or decrease of candidate set generation. So this research is focused on developing the new partition algorithm which decreases the execution time of the algorithm as size of the database increases. The methodology involved in the new partition algorithm is discussed in the next chapter.

When comes to dynamic databases, the border algorithm tries to reduce the I/O operations by using the promoted border concept. As this research is much focused on the static databases, the comparisons of performance of algorithms which deal with dynamic databases are not discussed in this section.

2.4 Summary

This chapter presents the literature review on the background and relevant research of Association rule mining and the algorithms used for mining of association rules. The following is the summary of the literature review:

- Association rule mining is a descriptive mining technique used for uncovering relations among data.

- A typical application of association rule mining is market-basket analysis, where the buying habits of the customers are analyzed to gain insight into the matters, such as "which items are most frequently purchased by customers".
- An association rule mining has two measures called the confidence and the support.
- The task of association rule mining is to discover all the rules that have support and confidence greater than or equal the user-specified minimum support and minimum confidence, respectively.
- A priori algorithm also called as level-wise algorithm is a popular algorithm used for mining frequent sets. It includes two phases called the candidate set generation and the pruning process.
- A priori algorithm requires number of database passes as equal to number of passes the algorithm runs for generating the frequent itemsets.
- Partition algorithm is based on the observation that the frequent sets are normally very few in number compared to the set of all itemsets.
- Partition algorithm breaks the transaction database into smaller segments such that each segment can be accommodated in the main memory.
- Incremental generation of association rules may be easier to perform by treating the current state of the database as one partition and treating the new entries as a second partition.
- As in practice, no transaction database is static, incremental approach will determine an intelligent use of the earlier computation of frequent sets, to find the

frequent sets of the whole database which includes both the old and new incremental part of the database.

- Incremental approach uses a concept called a Promoted border, in order to know whether an additional pass is required on the old part of the database for finding frequent sets of whole database.
- An algorithm called Border algorithm is developed for finding the frequent itemsets by incremental approach called the promoted border.

CHAPTER III

RESEARCH METHODOLOGY

In previous chapter, different approaches have been proposed to generate association rules effectively. These proposed approaches have their own advantages and disadvantages. In this chapter, the new partition approach is described in finer detail. The outline of this chapter is as follows: Section 3.1 discusses the methodology used in the new partition algorithm and emphasizes the changes made to the partition approach discussed in the previous chapter. Section 3.2 explains the generation and pre-processing of the datasets which will be used in the thesis. Section 3.3, discusses the performance of new partition algorithm. Finally, Section 3.4 gives a summary of this chapter.

3.1. Problem Specification

Most of the algorithms for discovering association rules require multiple passes over the database resulting in a large number of disk reads and placing a huge burden on the I/O subsystem. In order to reduce the burden on the I/O subsystem in the case of large databases, a new association rule mining algorithm, which uses both the Partition [AGG1998] [FAST1995] [AGS1996] and the Apriori approach [AGG1998] [FAST1995] [AGS1996] for calculating the frequent itemsets in a single pass over the database, is described below. This new association rule mining algorithm can also be used even when the database is growing intermittently (To see how this new proposed association rule mining algorithm works for databases which grows intermittently refer to future research (Chapter V)).

3.1.1 Proposed Extensions to Partition algorithm

This section discusses the approach that has been proposed for the partition algorithm useful for mining frequent itemsets.

The following notation is used in the remainder of this approach:

Table 3.1 Notations used for Partitioned Approach

Notation	**Meaning**
L^i	*Local Frequent Sets:* Set of Local Frequent Itemsets of partition i.
C_k^G	*Global Frequent Sets:* Set of global candidate k-Itemsets.
L_i^k	*Local Frequent Sets:* Set of local frequent k-Itemsets in partition i.

L^G	*Global Frequent Itemsets* Set of global frequent Itemsets.
$s(c)_{Tc}$	*Combined Support* Total support of candidate set c in all partitions.

This modified partition approach used for finding frequent itemsets in single pass over the database consists of two phases. The methodology involved in those two phases is described below:

Phase 1:

In this phase, the partition algorithm logically divides the database into a number of non-overlapping partitions. These partitions are considered one at a time and all frequent itemsets for that partition (L^i) are generated using the apriori algorithm (refer to section 2.2.1 in chapter 2). In addition, when taking each partition for calculating the frequent itemsets separately the local minimum support is set to 1. Thus, if there are n partitions, phase I of the algorithm takes n iterations. At the end of phase I, all the local frequent itemsets of each partition are merged to generate a set of all potential frequent itemsets. In this step, the local frequent itemsets of same lengths from all n-partitions are combined to generate the global candidate itemsets (C_k^G), and also those global candidate itemsets has there combined support (total support of itemset if that itemset is present in more than one partition) associated with it.

Phase II:

As the above generated global candidate itemsets have least possible frequent itemset of that partition because, during the generation of frequent itemsets using the apriori algorithm of each partition, the minimum local support was set to 1. So this phase just prune the itemsets from the global candidate itemsets list whose combined support ($s(c)_{Tc}$)(total support of an itemset in all the partitions) is less than the global minimum support. So by using the above approach the extra database pass which was needed in the phase II of the previous Partition approach (refer to section 2.3.2 in chapter 2) for calculating the support of global candidate itemsets is eliminated. So here the modified partition algorithm reads the entire database once during the Phase I. And also, partition sizes are chosen such that each partition can be accommodated in the main memory.

Below is the algorithm of modified partition approach:

```
P = partition_database(T); N = Number of partitions;

// Phase I
  for i = 1 to n do begin
  read_in_partition( $T_i$ in P )
  $L^i$ = generate all frequent itemsets of $T_i$ using a priori method in main
memory.
```

```
end
// Merge Phase
 for ( k = 2 ; $L_i^k \neq \Phi$ , i = 1,2,...,n ; k++) do begin
     $C_k^G = \bigcup_{i=1}^{n} L_i^k$
 end
// Phase II
$L^G = \Phi$;
 for each c $\in C^G$ do begin
     if $s(c)_{Tc} \geq \sigma$
     $L^G = L^G \cup \{s(c)\}$
   end
   Answer = $L^G$
```

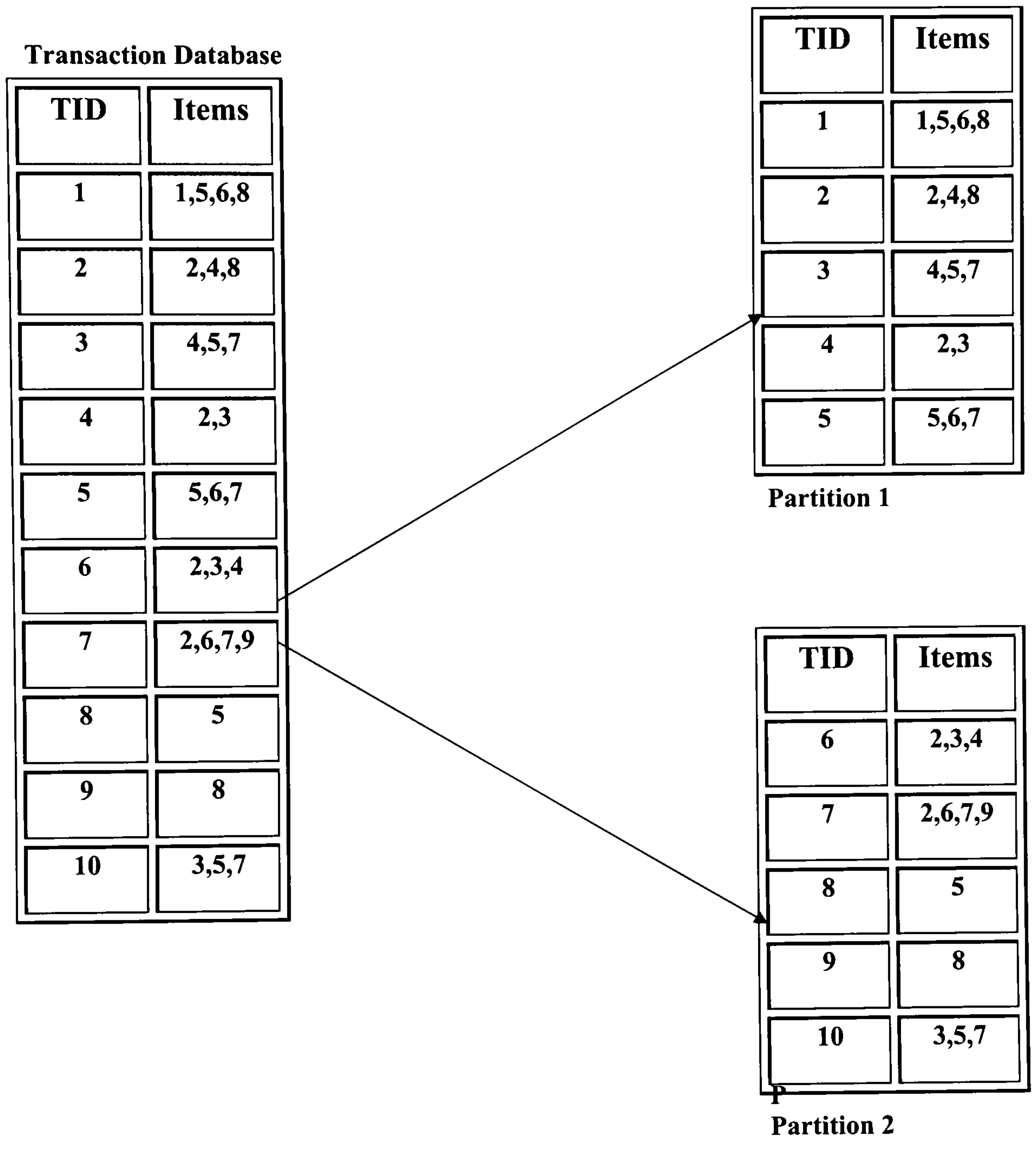
This above partition approach is based on the premise that the number of items in a single transaction is considerably smaller compared to the total items in the transaction database (i.e. total number of items placed in basket is less compared to total number of items available). In addition, it expects the support of frequent itemsets generated in a particular partition to be much high (more than 1). Therefore, for sufficiently large partition sizes, the number of local frequent itemsets is likely to be comparable to the number of frequent itemsets generated for the entire database. If the data characteristics are uniform across partitions, then large numbers of itemsets generated for individual partitions may be common.

Below is the example which shows the working of the above explained partition approach.

Here T, is the transaction database and A is the total items in transaction database A = { A1,A2,A3,A4,A5,A6,A7,A8,A9}, and T is divided into two partitions, partition 1 (transactions 1 to 5) and partition 2 (transactions 6 to 10). T is:

A1	A2	A3	A4	A5	A6	A7	A8	A9
1	0	0	0	1	1	0	1	0
0	1	0	1	0	0	0	1	0
0	0	0	1	1	0	1	0	0
0	1	1	0	0	0	0	0	0
0	0	0	0	1	1	1	0	0
0	1	1	1	0	0	0	0	0
0	1	0	0	0	1	1	0	1
0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	1	0
0	0	1	0	1	0	1	0	0

Table 3.2 Transaction Database [ARUN2003]



Transaction Database

TID	Items
1	1,5,6,8
2	2,4,8
3	4,5,7
4	2,3
5	5,6,7
6	2,3,4
7	2,6,7,9
8	5
9	8
10	3,5,7

TID	Items
1	1,5,6,8
2	2,4,8
3	4,5,7
4	2,3
5	5,6,7

Partition 1

TID	Items
6	2,3,4
7	2,6,7,9
8	5
9	8
10	3,5,7

P

Partition 2

Figure 3.2 Shows the (tid,item) format of the transaction database T.

And if here user specified minimum support, σ = 20%, i.e. an itemset is frequent if it supports at least 2 transactions in transaction database T (as transaction database T has 10 transactions, so 20% comes to 2). As the original implementation of modified partition algorithm uses the (tid,item) format as its input, the above figure shows the (tid,item) format of the transaction database T.

TID	Items
1	1,5,6,8
2	2,4,8
3	4,5,7
4	2,3
5	5,6,7

Table3.3 Partition 1

Here the frequent itemsets of the partition 1 will be generated using the apriori algorithm (refer to example in section 2.3.2 in chapter 2 for the procedure of generating the frequent itemsets using the apriori algorithm), here the user-specified local minimum support is 1.

L_1:={{1}→1,{2}→2,{3}→1,{4}→2,{5}→3,{6}→2,{7}→2,{8}→2,{1,5}→1,{1,6}→1, {1,8}→1, {2,3}→1,{2,4}→1,{2,8}→1,{4,5}→1,{4,7}→1,{4,8}→,{5,6}→2, {5,7}→2, {5,8}→1,{6,7}→1,{6,8}→1,{1,6,8}→1,{1,5,6}→1,{1,5,8}→1,{2,4,8}→1,{4,5,7}→1, {5,6,8}→1,{5,6,7}→1,{1,5,6,8}→1 }

TID	Items
6	2,3,4
7	2,6,7,9
8	5
9	8
10	3,5,7

Table 3.4 Partition 2

Here the frequent itemsets of the partition 2 will be generated using the apriori algorithm (refer to example in section 2.3.2 in chapter 2 for the procedure of generating the frequent itemsets using the apriori algorithm), here the user-specified local minimum support is 1.

L_2:={{2}→2,{3}→2,{4}→1,{5}→2,{6}→1,{7}→2,{8}→1,{9}→1,{2,3}→1,{2,4}→1, {2,6}→1,{2,7}→1,{2,9}→1,{3,4}→1,{3,5}→1,{3,7}→1,{5,7}→1,{6,7}→1,{6,9}→1,{ 7,9}→1,{2,3,4}→1, {2,6,7}→1,{2,7,9}→1,{3,5,7}→1,{2,6,9}→1,{2,6,7,9}→1 }

Then C, the candidate set is L1 $\cup$ L_2.

C:={{1}→1,{2}→4,{3}→3,{4}→3,{5}→5,{6}→3,{7}→4,{8}→3,{9}→1,{1,5}→1,{1, 6}→1,{1,8}→1,{2,3}→2,{2,4}→2,{2,6}→1,{2,7}→1,{2,8}→1,{2,9}→1,{3,4}→1, {3,5}→1,{3,7}→1,{4,5}→1,{4,7}→1,{4,8}→1,{5,6}→2,{5,7}→3,{5,8}→1,{6,7}→2, {6,8}→1,{1,6,8}→1,{1,5,6}→1,{1,5,8}→1,{2,4,8}→1,{2,3,4}→1,{2,6,7}→1,{2,6,9}

$\rightarrow 1, \{2,7,9\} \rightarrow 1, \{3,5,7\} \rightarrow 1, \{4,5,7\} \rightarrow 1, \{5,6,8\} \rightarrow 1, \{5,6,7\} \rightarrow 1, \{1,5,6,8\} \rightarrow 1,$

$\{2,6,7,9\} \rightarrow 1 \}$

In Phase II:

The sets which don't have user specified minimum support in the above candidate sets generated will be pruned.

As here the user-specified minimum support, $\sigma = 20\%$. i.e. the support of the candidate sets need to be at least 2 to be a frequent set.

Therefore the frequent itemsets of the transaction database T, with minimum support of 20% are

$L^G := \{\{2\} \rightarrow 4, \{3\} \rightarrow 3, \{4\} \rightarrow 3, \{5\} \rightarrow 5, \{6\} \rightarrow 3, \{7\} \rightarrow 4, \{8\} \rightarrow 3, \{2,3\} \rightarrow 2, \{2,4\} \rightarrow 2, \{5,6\} \rightarrow 2, \{5,7\} \rightarrow 3, \{6,7\} \rightarrow 2\}$

3.2 Data sets

For the purpose of implementing the above partition based association rule mining algorithm for finding the frequent itemsets, the retail market basket data set supplied by an anonymous Belgian retail supermarket store is used. These data sets were obtained from FIMI repository for frequent itemset mining (http://fimi.cs.helsinki.fi/data/).

The data used in this research are collected over the three non-consecutive periods. The first period runs from half December 1999 to half January 2000. The second period runs from May 2000 to the beginning of June 2000. The third and the final period run from the end of August 2000 to the end of November 2000. The total amount of receipts being collected during this period equals 88,163.

Each record in the above collected data set contains information about the date of Purchase (variable 'date'), the receipt number (variable 'receipt_nr'), the article number (variable 'article_nr'), the number of items purchased (variable 'amount'), the article price and the customer number (variable 'costumer_nr'). Over the entire data collection period, the super market store carries 16,470 unique items, but some of them only a seasonal basis, e.g. Christmas items. In total, 5133 customers have purchased at least one product in the super market during the data collection periods.

Other datasets provided at FIMI repository for frequent itemset mining (http://fimi.cs.helsinki.fi/data/) are dataset of traffic accidents which is obtained from the National Institute of Statistics for the region of Flanders for the period of 1991-2000. The traffic accident data contain a rich source of information on the different circumstances in which the accidents have occurred: course of the accident, traffic conditions, environmental conditions, road conditions. Other datasets are IBM- Artificial, BMS-POS which contain point-of-scale data from a large electronics retailer. The dataset contains transaction data of a customer purchasing different products at one time. The goal for this dataset is to find associations between product categories purchased by customer in a single visit to the retailer. Of all the datasets provided the dataset which contains the retail data of the Belgium supermarket is close to Market basket analysis problem, and as this data is real world data (not produced from some data generator), so this dataset is chosen for performance evaluation of the algorithms used in this thesis.

3.2.1. Data Pre-Processing

The data pre-processing procedure includes a data reformatting process and the discretization of numerical attributes. The original data format of the data sets is in RDBMS (tables, rows, columns) format. However, the association rule algorithm use a different input data format, which requires a data format transformation, the below are the two data examples:

```
36 37 38 39 40 41 42 43 44 45 46
38 39 47 48
```

Here above are the two different transactions, each having different items in it. Format transformation was done by separating each item in a single transaction by space and two transactions by line separation in a text-editor (for example, Notepad).

Before the input is fed to the mining algorithm the input is checked for (tid, item) format. On completion of mining, the results are remapped to their original values. Since the time taken for mapping, rule generation and re-mapping the results to their original description is relatively insignificant, they are not reported in this thesis.

3.2.2 Implementation of Advanced Partition Approach

The advanced partition algorithm is implemented by modifying the original partition algorithm proposed by

Source code of the previous partition algorithm is obtained from (http://www.helsinki.fi/~holler/datamining/algorithms.html, March 16, 2004). The advanced partition approach produces the frequent itemsets in single pass over the database, where as, the previous partition approach uses two passes over the database to find the frequent itemsets. To do this, a few additional modules, including the changes in the Apriori module and partition module are created. The system is implemented as a object oriented program in the Java programming language.

3.3 Discussion on Different Algorithms

As discussed in the previous chapter, the two main aspects of the association rule mining algorithms are the candidate generation process and I/O operations. The performance analysis between Apriori algorithm and partition algorithm in previous chapter showed that the main aspect on which performance of the association rule mining algorithm depends is the I/O operations. It clearly showed (see section 2.4 in chapter 2) that performance of partition algorithm is much better than the Apriori algorithm when the database size increases. In order to reduce the I/O operations much more effectively, the newly proposed partition algorithm, calculates the frequent itemsets is a single pass which reduces the I/O operations and increases the performance of the algorithm have been discussed in the previous section.. So the chapter 4 gives the results of the performance analysis of the newly proposed partition approach and the previously discussed partition and Apriori algorithms.

3.4 Summary

Below is the summary of all the issues discussed in this chapter:

- A modified approach of a partition algorithm is used for finding the frequent itemsets in single pass over the database.
- The modified partition approach reduces the burden on the I/O subsystem in case of large databases.
- For implementing the modified approach the retail market data set supplied from a anonymous Belgian retail supermarket store is used.
- The data pre-processing procedure includes a data reformatting process and the discretization of numerical attributes.
- The transaction database is converted into (tid, item) format, as the original implementation of the partition approach needs (tid, item) format as its input format.
- The modified approach uses the Apriori algorithm for finding the frequent itemsets of each partition separately.
- The modified partition approach uses two phases for calculating the frequent itemsets.
- This new approach reads the database once in the phase 1 only.
- The minimum local support used in new partition algorithm for finding the frequent itemsets using the Apriori algorithm is 1.
- The new approach is based on the premise that number of items in a transaction is quite less compared to total number of items in the transaction database.

- The new partition approach is efficient when the local support of each frequent itemset in a partition is much higher than 1.
- If the data characteristics are uniform across partitions, then large number of itemsets generated for individual partitions may be common.
- Partitions should be made as such they will be accommodated in the main memory.
- The new partition approach eliminates the need of extra pass over the transaction database to find the frequent itemsets, which is done in the partition approach discussed in the chapter 2.
- The global candidate itemsets generated in the phase 1 of new partition approach has all the least possible frequent itemset of the transaction database, as the local minimum support is set to 1 when calculating the frequent itemsets of each partition separately.
- The new partition approach is implemented as a object oriented program in the java programming language.

CHAPTER IV

RESEARCH RESULTS

This chapter presents and analyzes the results of the experiments conducted in this thesis. Section 4.1 deals with the description of the datasets and explains how the results are summarized. Section 4.2 compares the performance of the new partition approach with previously discussed algorithms. Section 4.3 summarizes the chapter.

4.1 Data Sets

As discussed in the previous chapters, the two important and common steps of association rule mining algorithms are candidate generation and I/O operations. The initial candidate set generation is the key issue to improve performance of the frequent set discovery algorithms. Another performance measure is the amount of data that has to be scanned during the discovery of the frequent itemsets. From the comparisons of the performances of the Apriori and Partition algorithms in chapter 2, it is evident that the most significant difference in execution times comes with the increase or decrease of the number of transactions (I/O operations) than the increase or decrease of candidate set generation. In order to improve the performance of association rule algorithms when the number of transactions increases, the new partition approach which calculates the frequent itemsets in a single database pass is proposed as given in chapter 3.

To measure the performance of the New Partition approach with the previously discussed Apriori and Partition algorithms, the datasets from the FIMI repository for frequent itemset mining (http://fimi.cs.helsinki.fi/data/) are used. As given in the chapter 2, the nomenclature of these datasets is of the form "TxxDzzzK", where "xx" denotes the average number of items present per transaction and "zzzK" denotes the total number of transaction on "K"(1000's). The experiments are performed on a machine running Microsoft Windows XP with 786 MB of RAM and a 1.6 GHz Pentium 4 Processor. Each experiment has been performed 4 times. The values from the first run are ignored so as to avoid the effect of the previous experiment and other database setups. The average of the next 3 runs is taken and used for analysis. This is done so as to avoid any false reporting of time due to system overload or any other factors. For most of the experiments, the percentage of difference of each run is less than one percent.

4.2 Comparisons of Performances of Different Algorithms

To compare the performance of the New Partition algorithm described in the previous chapter with the Apriori and Partition algorithm discussed in the chapter 2 by using the data sets described above, three different scenarios are considered:

- Scenario 1: Performance of the New Partition algorithm when the average number of items per transaction increases.
- Scenario 2: Performance of the New Partition algorithm when the data set size is small and the number of items per transactions are also small.
- Scenario 3: Performance of the New Partition algorithm when the size of the data set is large and candidate itemsets are also large.

The following subsections perform the above specified scenarios separately:

4.2.1 Scenario 1

To compare the performance of the new partition algorithm when the number of items per transaction increases, the following datasets are chosen:

Name	|T|	|D|	Size in Megabytes
T9D80K	9	80K	3.5
T13D80K	13	80K	4

Where |T| is the average number of items present in the transaction and |D| is the number of transactions in a dataset (in 1000's).

Below figures 4.1 and 4.2 shows the execution times of Apriori and Partition and New Partition algorithms under different minimum support.

T9D80K - Figure 4.1 Performance T9D80K			
Minimum Support (%)	Apriori (Seconds)	Partition (Seconds)	Partition 1(Seconds)
2	76	102	449
1.5	88	116	450
1	103	141	450
.75	137	177	448
.5	154	204	451
.33	172	231	450

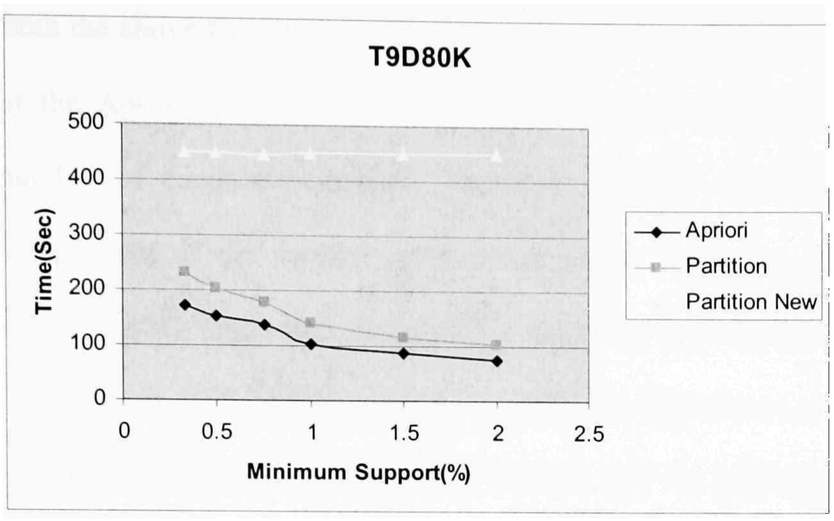


Figure 4.1 Performance T9D80K

T13D80K - Figure 4.2 Performance T13D80K			
Minimum Support (%)	Apriori (Seconds)	Partition (Seconds)	Partition 1(Seconds)
2	109	111	509
1.5	110	117	510
1	115	124	509
.75	118	144	508
.5	128	179	509
.33	148	288	509

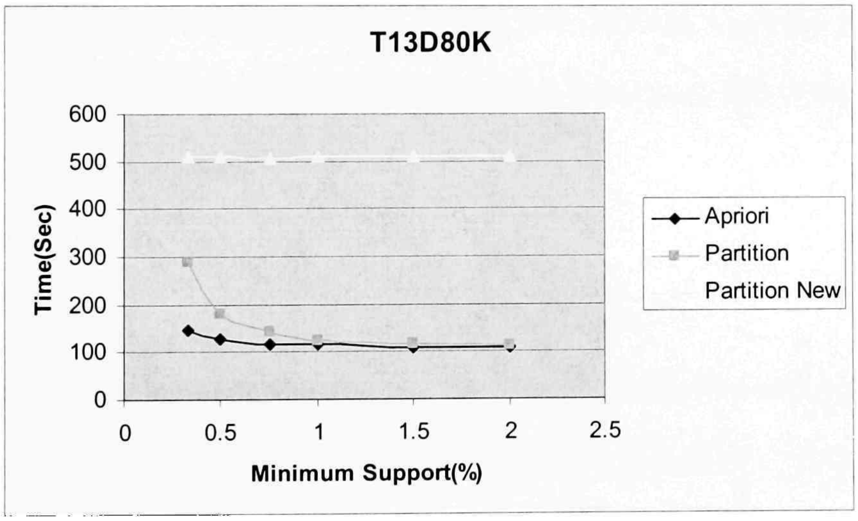


Figure 4.2 Performance T13D80K

Both the above figures show that as the minimum support decreases, the execution times of the Apriori and Partition algorithms increase because of the increase in the total number of candidate itemsets. The New Partition algorithm execution however is independent of the increase or decrease of the minimum support (because the local support for the itemsets is treated as 1 irrespective of the global support). Figures 1 and 2, it is evident that the New Partition approach performance is good when the average numbers of items in the transaction are less and the performance of the new partition approach is way less than the both the partition and Apriori algorithm when the database size is small and even the average number of items in the transaction change.

Summary: Both the partition and Apriori algorithms execution times beat the new partition approach almost by a magnitude of three when the database size is small and the candidate generation increases. In other words, the new partition approach performance will be good when the average number of items per transaction is less. So the next scenario sees the performance of the new partition approach when the database size changes under the different minimum support and when the average numbers of items per transaction are less.

4.2.2 Scenario 2

To analyze the performance of the New Partition algorithm when the database size increases and the minimum support decreases, the following datasets are chosen:

Name	|T|	|D|	Size in Megabytes
T9D90K	9	90K	4

T9D180K	9	180K	8
T9D360K	9	360K	16

Table 2

Where |T| are the average number of items present in the transaction and |D| are the number of transaction in a datasets (in 1000's).

Figures 4.3, 4.4, 4.5 shows the execution times of all the three algorithms for different database sizes and having minimum percentage support of 2, 1, and .50 respectively. Below tables show the values corresponding to figures 4.3, 4.4, 4.5.

SUPPORT 2% - Figure 4.3 Performance Support 2%			
DATASETS	Apriori (Seconds)	Partition (Seconds)	Partition 1(Seconds)
T9D90K	70	115	500
T9D180K	133	230	998
T9D360K	267	458	1997

SUPPORT 1% - Figure 4.4 Performance Support 1%			
DATASETS	Apriori (Seconds)	Partition (Seconds)	Partition 1(Seconds)
T9D90K	93	160	502
T9D180K	180	320	999
T9D360K	362	638	2000

SUPPORT .50% - Figure 4.5 Performance Support .50%			
DATASETS	Apriori (Seconds)	Partition (Seconds)	Partition 1(Seconds)
T9D90K	140	257	500
T9D180K	280	510	1000
T9D360K	560	1000	1998

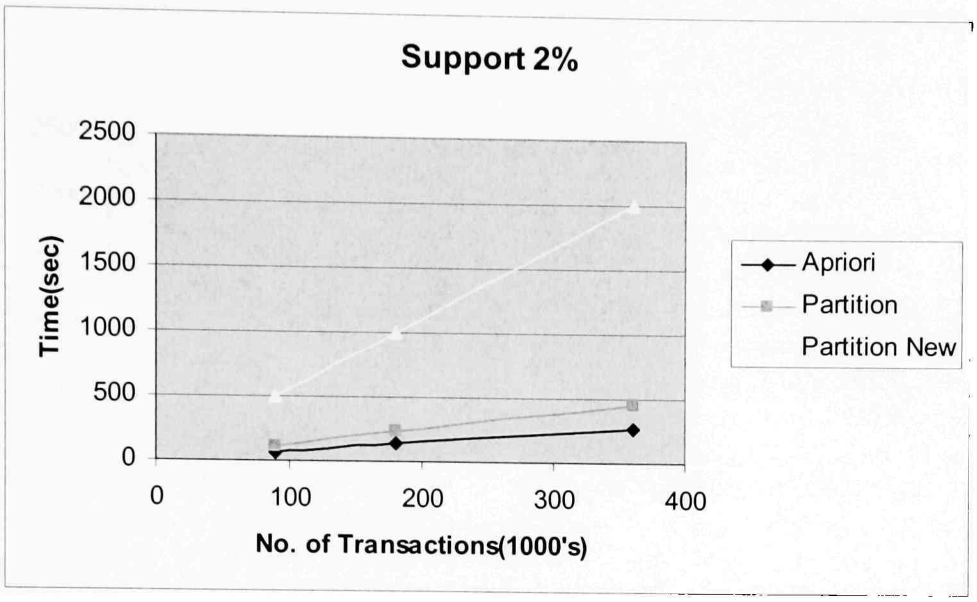


Figure 4.3 Performance Support 2%

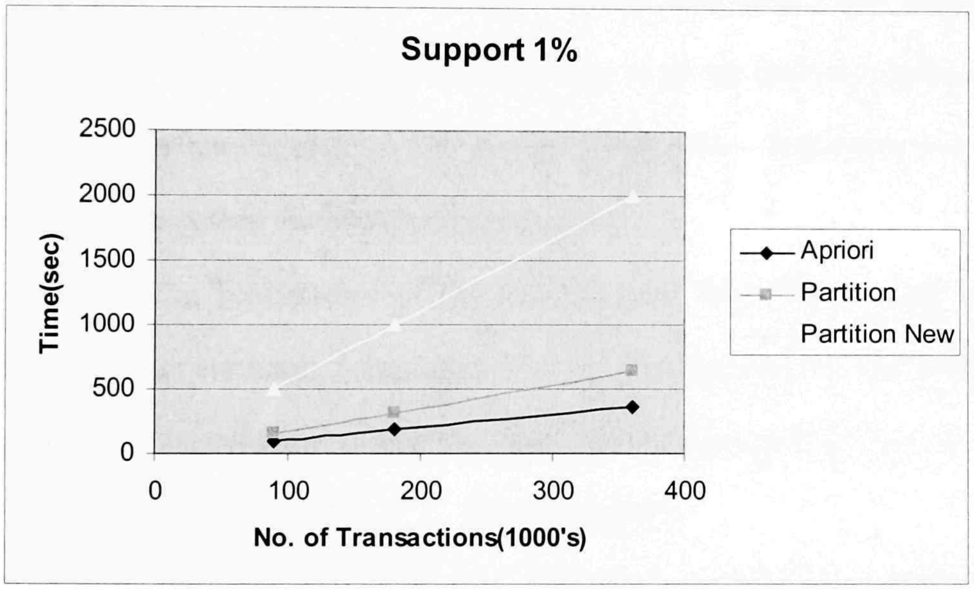


Figure 4.4 Performance Support 1%

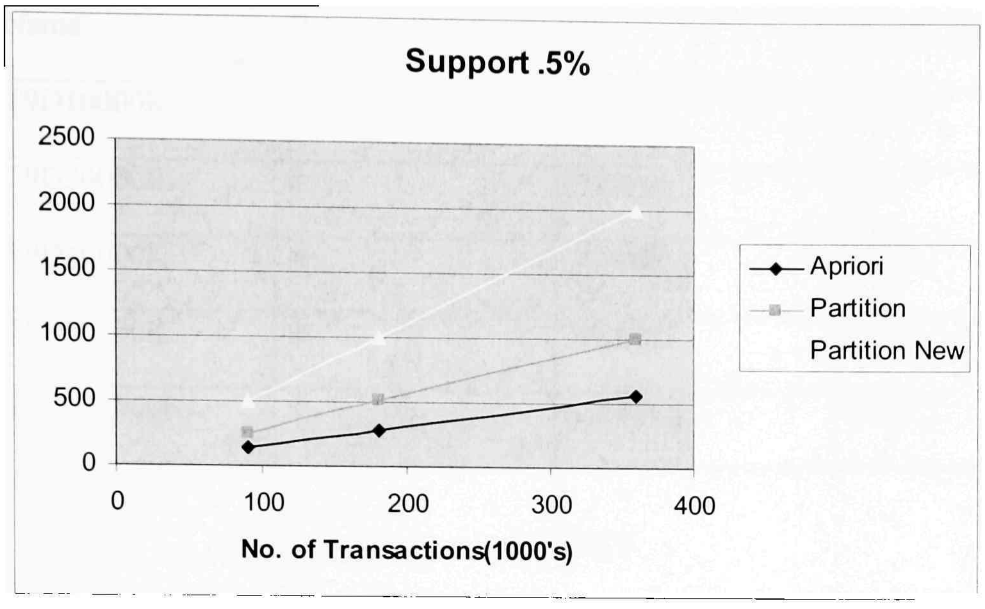


Figure 4.5 Performance Support .5%

Figures 3, 4, and 5 show that the execution time of the New Partition algorithm increases as the size of the database increases, as do the execution times of the Apriori and the Partition algorithms. The Apriori and Partition algorithms still have smaller execution times than the New Partition algorithm.

Summary: The performance of the new partition algorithm is much less when the database sizes are small, comparatively the performance of the new partition approach increases as the minimum support decreases and the database size increases.

4.2.3 Scenario 3

To compare the performance of the New Partition algorithm when the database size is large and the minimum support decreases, the following datasets are chosen:

Name	|T|	|D|	Size in Megabytes
T9D10000K	9	10000K	250
T9D20000K	9	20000K	500
T9D40000K	9	40000K	1000
T9D80000K	9	80000K	2000
T9D160000K	9	160000K	4000

Table 3

Where |T| is the average number of items present in the transaction and |D| is the number of transaction in a datasets (in 1000's).

Figures 4.6, 4.7, 4.8, and 4.9 show the execution times of all the three algorithms for different database sizes and having minimum percentage support 2, 1, .50, .25 respectively. As the number of transactions are large and are difficult to represent on the graph, the size of the database in Megabytes(MB) is used. Below tables show the values corresponding to figures 4.6, 4.7, 4.8, and 4.9.

SUPPORT 2% - Figure 4.6 Performance Support 2% (Large Databases)			
DATASETS (MB)	Apriori (Hours)	Partition (Hours)	Partition 1(Hours)
250	0.83	1.3	5
500	1.6	2.9	10
1000	3.3	6.6	20
2000	6.6	14.9	40
4000	13.2	34	80

SUPPORT 1% Figure 4.7 Performance Support 1% (Large Databases)			
DATASETS (MB)	Apriori (Hours)	Partition (Hours)	Partition 1(Hours)
250	1.1	1.8	5
500	2.3	4.1	10
1000	5	9.3	20
2000	10.5	21.1	40
4000	22.1	48.6	80

SUPPORT .50% Figure 4.8 Performance Support .50% (Large Databases)			
DATASETS (MB)	Apriori (Hours)	Partition (Hours)	Partition 1(Hours)
250	1.7	2.8	5
500	3.6	6.4	10
1000	7.7	14.4	20
2000	16.2	32.2	40
4000	34	74.5	80

SUPPORT .25% - Figure 4.9 Performance Support .25% (Large Databases)			
DATASETS (MB)	Apriori (Hours)	Partition (Hours)	Partition 1(Hours)
250	3.4	3.7	5
500	7.2	8.5	10
1000	15.2	19.1	20
2000	32	43	40
4000	68	98	80

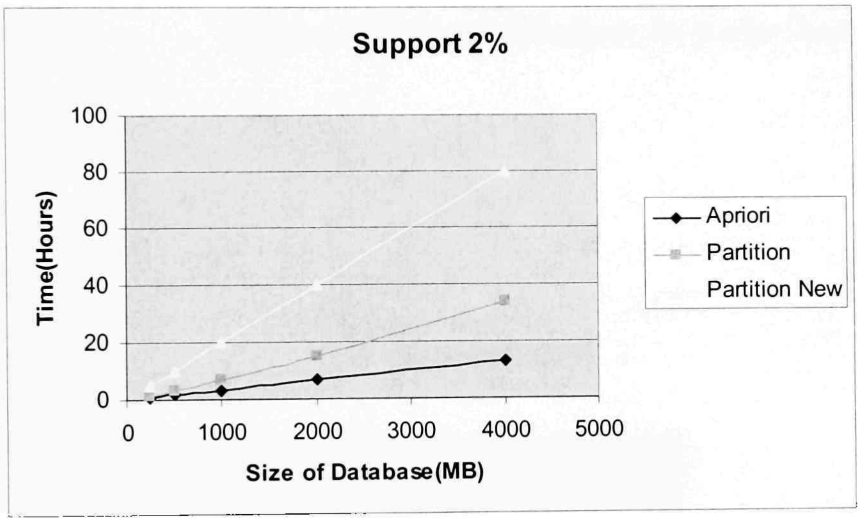


Figure 4.6 Performance Support 2% (Large Databases)

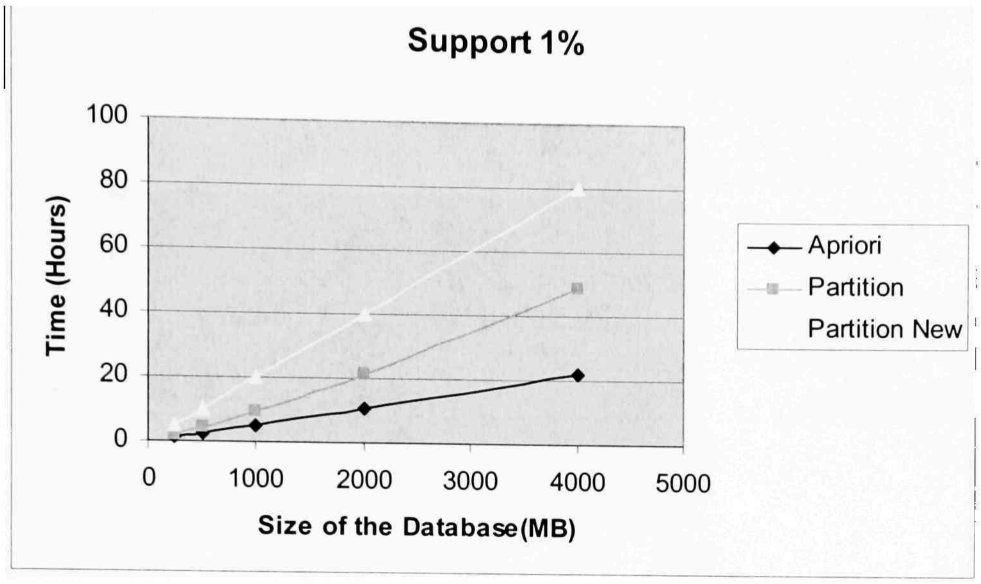


Figure 4.7 Performance Support 1% (Large Databases)

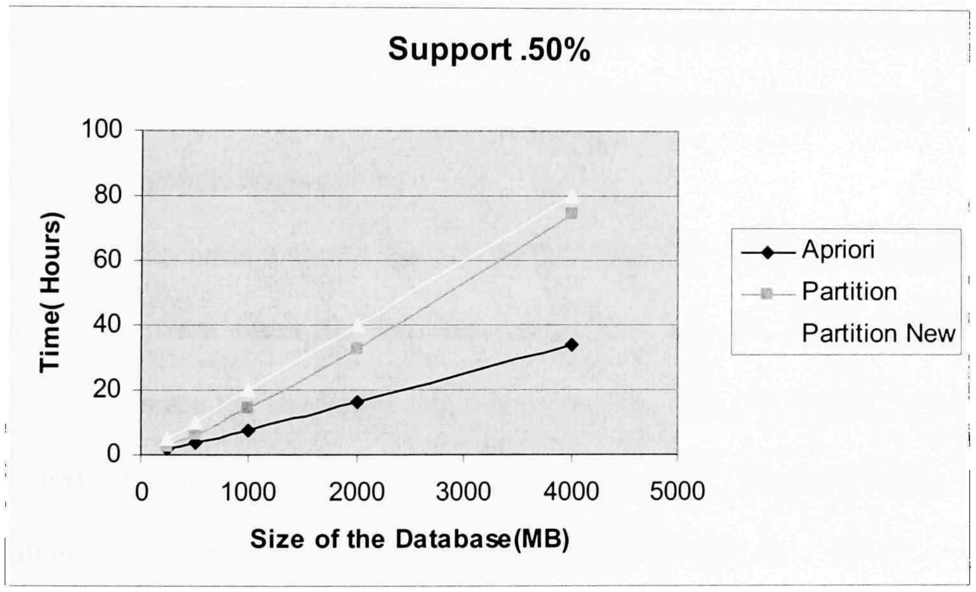


Figure 4.8 Performance Support .50% (Large Databases)

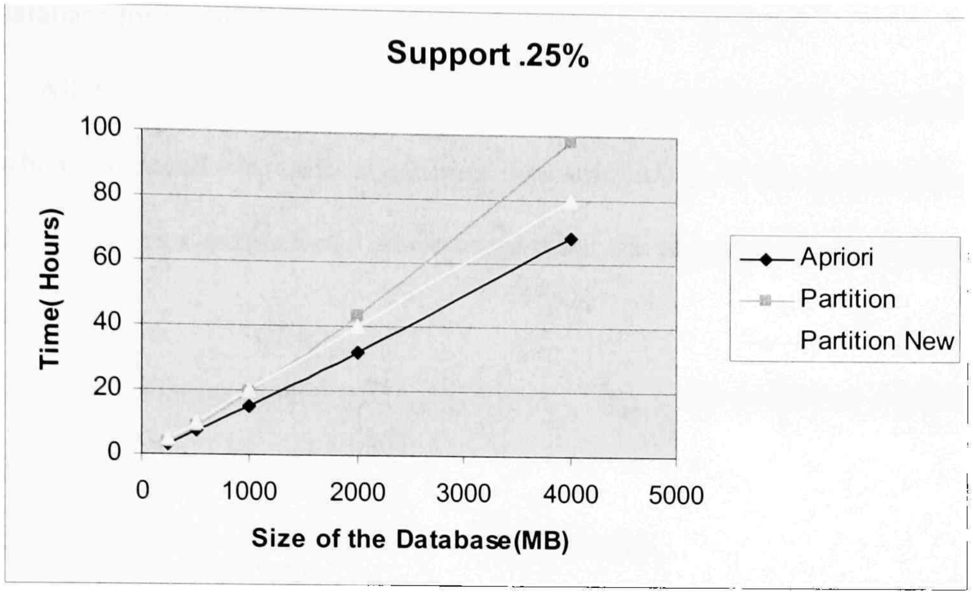


Figure 4.9 Performance Support .25% (Large Databases)

As seen in the above figures, The New Partition algorithm's execution time eventually becomes better than the execution times of the partition algorithm and almost close to the execution time of the Apriori algorithm as the database size increases and the minimum support decreases.

Summary: Scenario 3 shows that new partition algorithm will beat the performance of the other algorithms when the database size is huge and the minimum support decreases (which increase the candidate set generation). The new partition approach will beat the performance of the normal partition algorithm when the extra time needed for the candidate sets generation in new partition algorithm than the normal partition algorithm will be less than the time needed for normal partition algorithm to scan the whole database for pruning of the candidate sets.

All the above three scenarios shows the performances the new partition approach when compared with other algorithms, the conclusions of the experiments conducted and the difficulties encountered while performing the experiments are discussed in the next chapter.

The following section will summarizes all the above conducted experiments and their results:

4.3 Summary

The New Partition algorithm proposed in Chapter 3 with the goal of mining frequent itemsets using a single database pass, is used in this chapter and its performance is compared with the Apriori and Partition algorithms discussed in chapter 2 under three different scenarios. After conducting the performance analysis of the New partition algorithm, some issues are evident:

- The New Partition algorithm's performance was poor as compared to Apriori and Partition algorithm when database size is small and the average number of items increase.
- The New Partition algorithm's performance was better when the database size increases and the average number of items per transaction are lower.
- The New Partition algorithm's performance is independent of the minimum support, it only depends on the size of the database.

- The New partition algorithm is better than the performance of the Partition algorithm when the database size is large and support.
- When the extra time taken for candidate generation by new partition algorithm than the time taken for candidate generation by the normal partition algorithm is less than the time taken for the database pass in the pruning stage of the normal partition algorithm, the new partition algorithm beats the execution time of the normal partition algorithm.
- As minimum support decreases and the database size increases, the New Partition algorithm moves closer and eventually gets better than Apriori and Partition algorithms.

CHAPTER V

CONCLUSIONS AND FUTURE WORK

The goal of this thesis is to design a new association rule mining algorithm which uses the partition approach for mining the frequent itemsets in a single pass over the database. In order to achieve this goal, the methodology described in Chapter 3 is used to implement the new association algorithm. Chapter 4 presents the performance analysis of the new association algorithm and its results are compared with the existing Apriori and Partition algorithm. The next section summarizes the conclusions drawn from the results discussed in Chapter 4, while the following section suggests possible directions for future work.

5.1 Conclusions

As explained in Chapter 2, the performance analysis of the Apriori and Partition algorithms explains that the most significant difference in execution times of association rule algorithms comes with the increase or decrease in the number of transactions (I/O operations) than the increase or decrease of candidate sets. Because the major performance difference is with the database size, the partition algorithm is modified to take one database pass to form the New Partition algorithm described in chapter 3 and used to explain in chapter 4. Chapter 4 presents the results of experiments detailing the performance of New Partition algorithm.

From the results presented in chapter 4, it is evident that as database size increases enormously, the New Partition algorithm beats the execution times of both the Apriori and the Partition algorithm. Taking the results from the three scenarios on which performance analysis is performed in previous chapter there are many conclusions that can be drawn from those results:

- Scenario 1: Performance of the New Partition algorithm when the average number of items per transaction increases.

Results: Performance of the New Partition algorithm is too low when compared to existing apriori and partition algorithm.

Conclusion: The new partition algorithm whose frequent itemsets are calculated by setting the local support as 1, which takes lot of time generating the candidate sets than the normal partition algorithm whose local support is dependent on the global support. As analysis is performed by changing the average number of itemsets per transaction, the new partition algorithm's performance will decrease when the average items per transaction increase because as the average items per transaction increase, the number of candidate sets generated will also increase, which in turn makes the new partition algorithm less efficient. Therefore to get the maximum performance of the new partition algorithm, the dataset should have less number of items per transaction.

- Scenario 2: Performance of the new partition algorithm when the data set size is small and the number of items per transactions are less.

Result: The performance of the new partition algorithm is less than the apriori and the partition algorithm when the database size is small.

Conclusion: The new partition algorithm eliminates the need of scanning the database in the pruning step as it uses the local support as 1, which gets all the possible frequent sets. When the database size is small the time taken for the database scan for pruning is less than the extra time needed for the candidate generation by setting local support to 1 than the local support depending on the global support. Therefore the new partition algorithm should not be used when the database size is small.

- Scenario 3: Performance of the new partition algorithm when the size of the data set is huge and the candidate generation increases.

Result: The performance of the partition algorithm beats both Apriori and the partition algorithm as the database size increases.

Conclusion: In this scenario, the database size is huge, and the performance of the new partition algorithm will come close to partition and Apriori algorithm as the database size increase, and when the database size is so huge as the time taken for the scanning the database is more than the extra time needed for the candidate generation by setting local support to 1 than the local support depending on the global support. Therefore the new partition algorithm will be used when the database size is huge.

From all the above conclusions it is evident that new partition algorithm will be effective when the database size is huge and the average number of items per transaction are less.

The following section will explain the way the future research can be done to increase the performance of the association rule mining algorithms.

5.2 Future Work

The New Partition algorithm may be extended to work for databases that grow intermittently as follows.

- The new transactions are treated as a new partition.
- The results of the candidate generation step before the pruning step of the existing database must be stored.
- The candidate sets are generated on the new partition.
- The candidate sets of existing and new partitions are merged together.

As the size of the partitions in the partition algorithm depends upon the size of the main memory, as the main memory increases the size of the partition size also increases, which will increase the performance of the partition algorithm. If the partition algorithm performance is less dependent on the main memory, then the partition algorithm can be efficient even on the computers having less main memory.

Association rule algorithms may be implemented with the Ant Colony Optimization (ACO) [ANT1999] technique. One of the ideas behind using the ACO technique in mining association rules is to improve performance when the database is not static. The main underlying idea, loosely inspired by the behavior of real ants, is that of a parallel search over several constructive computational threads based on local problem data and on a dynamic memory structure containing information on the quality of previously obtained result. The collective behavior emerging from the interaction of the different search threads has proved effective in solving combinatorial optimization problems. A set of computational concurrent and asynchronous agents (a colony of ants) moves through states of the problem corresponding to partial solutions of the problem to solve. They move by applying a stochastic local decision policy based on two parameters, called trails and attractiveness. By moving, each ant incrementally constructs a solution to the problem. Using the trial and a heuristic function used in the Ant Colony algorithm, the scanning of the whole existing database will be eliminated in order to find the frequent sets of the whole database.

Applying the ACO-neural network to an association rule mining problem is, however, an unexplored, yet promising research area considering the wide application of neural networks in data mining tasks.

BIBLIOGRAPHY

[ARUN2003] Arun K Pujari (2003). *Data Mining Techniques (Edition 5)*: Hyderabad, India: Universities Press (India) Private Limited.

[MARG2003] Margatet H. Dunham (2003). *Data Mining, Introductory and Advanced Topics*: Upper Saddle River, New Jersey: Pearson Education Inc.

[JIAW2004] Jiawei Han (2004), *Data Mining, Concepts and Techniques*: San Francisco, CA: Morgan Kaufmann Publishers.

[ABRAH2002] Abraham Silberschatz (2002), *Database System Concepts* (Edition 4): New York, NY: McGraw-Hill Companies, Inc.

[MET2000] Mehmet M. Dalkilic (2000), *Introduction and Motivation to Data mining*, Ph.D. Thesis, 2000

[AGG1998] Aggarwal charu, and Yu Philip. Mining large itemsets for association rules. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 21, no. 1, March 1998.

[AGA1993] Agarwal R., Imielinski T., and Swami A. Mining associations between sets of items in massive databases. *In Proceedings of the ACM SIGMOD International Conference on Management of Data*, Washington D.C., May 1993, pp. 207-216.

[FAST1995] Agrawal R., Mannila H., Srikanth R., Toivonen H.. Fast Discovery of association rules. *Advances in Knowledge Discovery and Data Mining*. Chapter 12, AAAI/MIT Press, 1995.

[PAR1996] Agrawal R. and Shafer J.C. Parallel mining of association rules: Design implementation and experience. *IEEE Transactions on Knowledge Data and Engineering*, December 1996, 8(6): pp. 962-969

[AGS1996] Agrawal R., and Srikanth R. Fast algorithms for mining association rules. *Proceedings of the 20th International Conference on Very Large Database*, Santiago, 1994.

[ALS1997] Alsabti K., Ranka S., and Singh V. A one-pass algorithm for accurately estimating quantiles for disk-resident data. In *proceeding of* VLDB'97.

[AYA1999] Ayan N.F., Tansel A.U., and Arkun E. An efficient algorithm to update large itemsets with early pruning. In *Proceeding of the 5th International Conference on Knowledge Discovery and Data mining* (KDD' 99), San Diego, California, August 1999.

[MUL1998] Agarwal S., Agarwal R., Deshpandade P.M., and Gupta A. *On the computation of multidimentional aggregates*. VlDB, 1998.

[BAR 1998] Barbara D. Special Issue on Mining of Large Datasets; *IEEE Data Engineering Bulletin*, 21 (1), 1998.

[BAY1999] Bayardo R.J. Jr., Agarwal R., and Gunopulos D. Constraint-based rule mining in large, dense databases. In *Proceeding of the 15th International Conference on Data Engineering*, Sydney, Australia, March 1999.

[HIPP1998] Hipp J., Myka A., Wirth R., and Guntzer U. A new algorithm for faster mining of generalized association rules. In *Proceeding of the 2nd European Symposium on Principles of Data Mining and Knowledge Discovery* (PKDD'98) Nates, France, September 1998.

[TOM1999] Tom Brijs and Gilbert Swinnen and Vanhoof K., and Wets G. *The use of Association Rules for Product Assortment Decision: A Case Study*, pp: 254-260,1999.

[ANT1999] Marco Dorigo, Gianni Di Caro, The Ant Colony Optimization Meta-Heuristic. *In D. Corne, F. Glover, editors, McGraw-Hill*, ,1999.